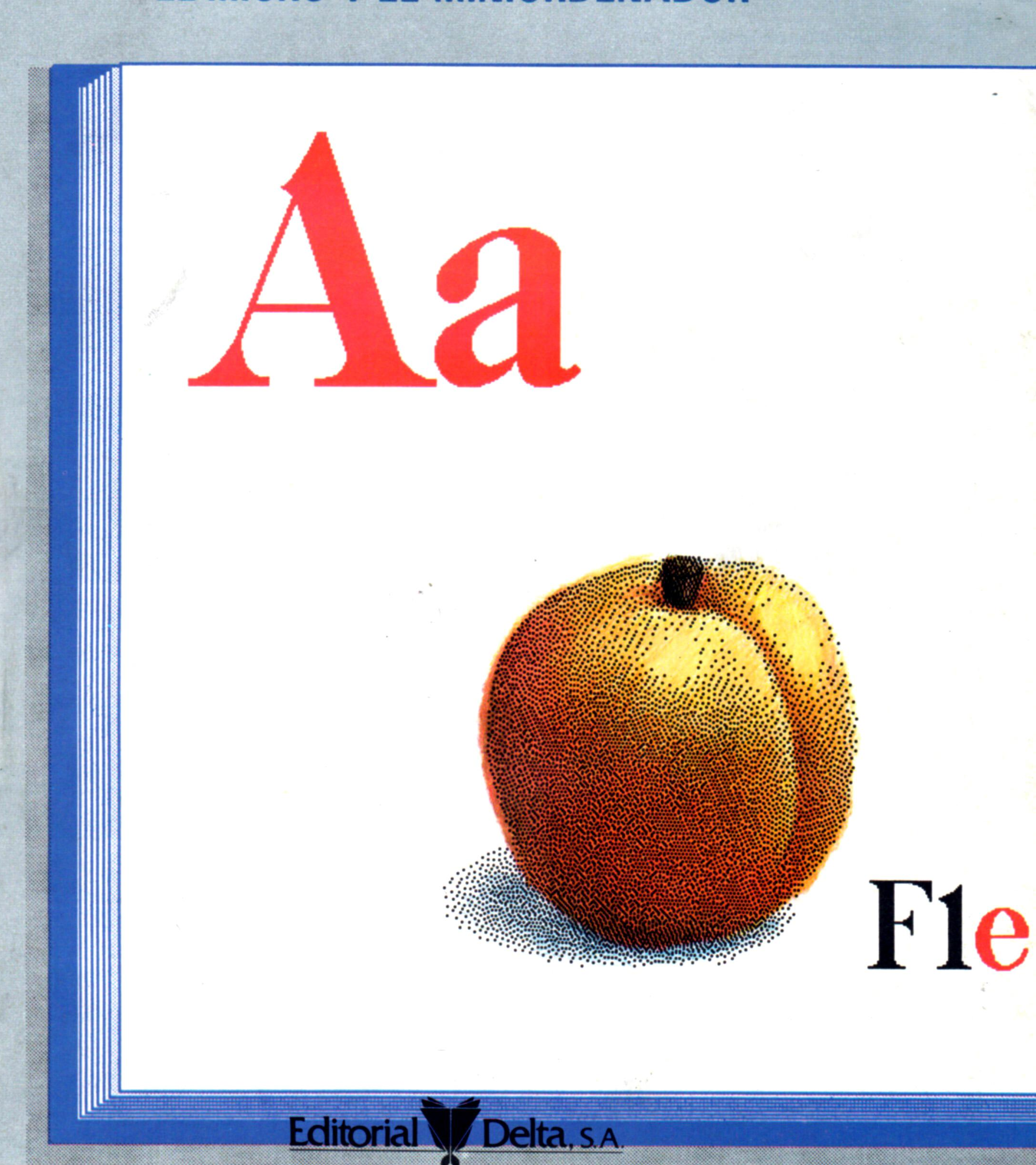
175 PTAS

# mi computer

94

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR




Editorial Delta, S.A.

# MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VIII-Fascículo 94

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.


ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 138511
Impreso en España-Printed in Spain-Noviembre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

**Servicio de suscripciones y atrasados** (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

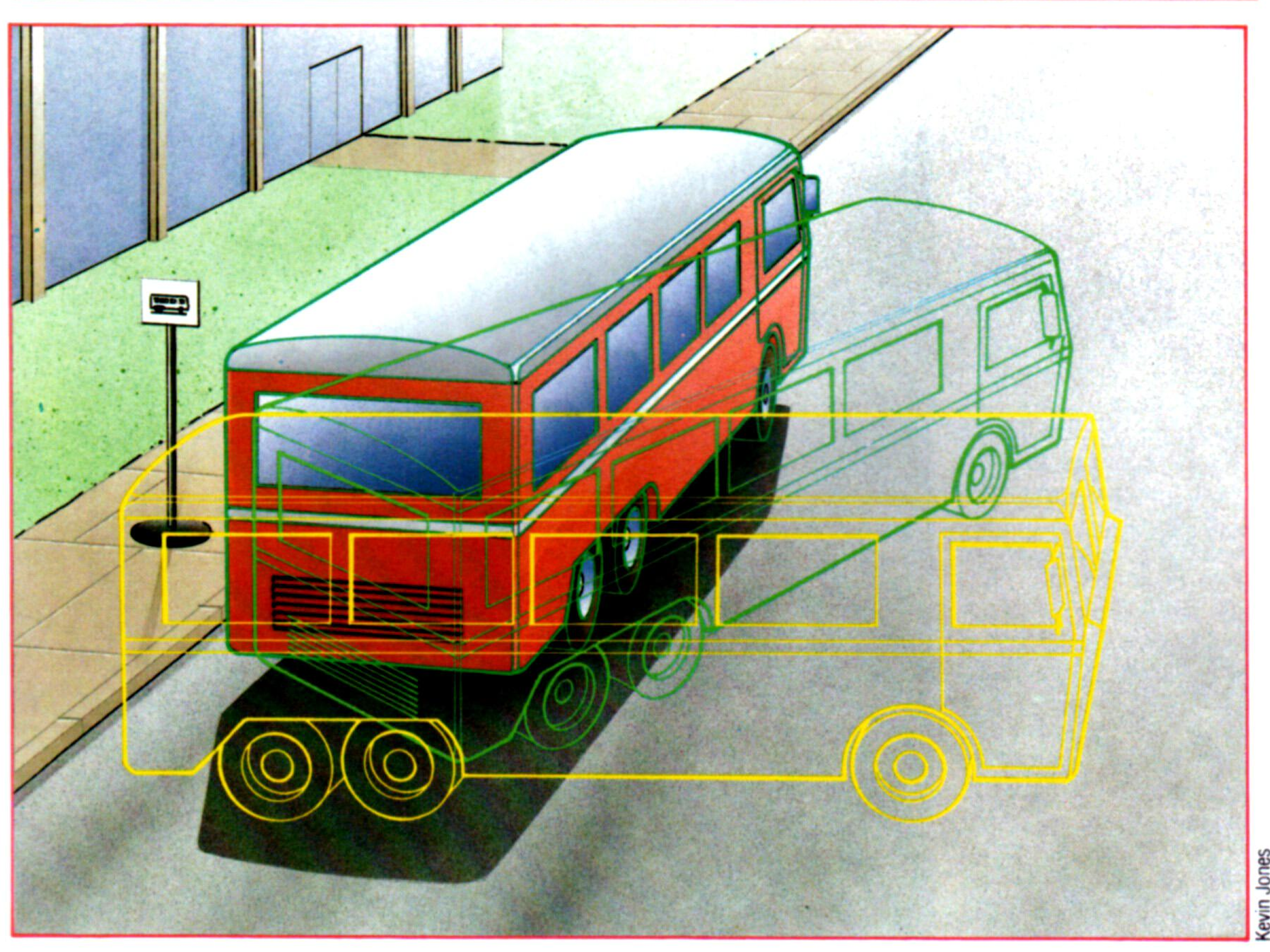
**No se efectúan envíos contra reembolso.**





# Conocimiento visual

**Al programar un ordenador para que «comprenda» lo que ve, es necesario diferenciar lo que se ve de lo que se conoce**



Kevin Jones

**Vista de lince**
Algunos sistemas de reconocimiento de patrones adoptan un enfoque *de arriba-abajo,* en el que se explora un patrón o escena en busca de la presencia de un objeto determinado. Sobre la pantalla se proyecta una representación esquemática del objeto en diversas orientaciones hasta que, si el objeto está presente, se encuentra una proyección que encaja en la silueta de la figura real. En una escena tridimensional, como la que vemos aquí, es muy grande la cantidad de posibles proyecciones que se podrían hacer de un modelo esquemático del autobús antes de hallar la correcta

Dicen que «la belleza está en los ojos del que mira»; pero sería mejor decir que la belleza está en el cerebro del perceptor o, con mayor precisión, en una compleja cadena de procesos nerviosos que comienza en el fondo de la retina y termina en algún lugar de la corteza occipital del cerebro. Es esta cadena de eventos lo que los fisiólogos intentan comprender y lo que los expertos en robótica intentan, en cierto sentido, imitar.

La percepción visual es un componente tan integral de la forma en que comprendemos nuestro entorno, que con frecuencia decimos «Ya veo» cuando en realidad queremos significar «Ya comprendo». La comprensión es realmente la clave del desafío que representa la visión por ordenador. De alguna forma, el ordenador debe tomar la información que le proporciona una cámara, u otro dispositivo sensible a la luz, y entender lo que esa información le dice sobre el estado de su entorno. Adquirir la información es fácil: el problema es cómo interpretarla.

El proceso de transformar imágenes en significados consta de tres etapas principales:

1. *Proceso de la imagen:* Convertir una imagen difusa o distorsionada en una más nítida. (La solución de esta tarea es relativamente simple.)
2. *Reconocimiento de patrones:* Detectar la presencia o la ausencia de rasgos u objetos significativos. (Esto es más complicado.)
3. *Comprensión de la imagen:* Determinar lo que está sucediendo en el mundo real. (Un problema muy complejo.)

Aunque la auténtica visión por ordenador (etapa 3) todavía no se ha alcanzado, de las dos etapas iniciales sí se han obtenido resultados útiles.

## Reconocimiento de patrones

El reconocimiento de patrones es una tarea fascinante, si bien nos resulta difícil «ver» las dificultades que entraña. Los reconocedores de patrones clasifican las imágenes emparejándolas con un conjunto limitado de alternativas, tales como las letras del alfabeto en el caso de los sistemas de reconocimiento de caracteres ópticos. Típicamente, extraen características mediante el examen de partes de la imagen digitalizada (pequeños grupos de pixels). El sistema puede entonces reconocer y clasificar patrones basados en la presencia o la ausencia de ciertas características distintivas.

El reconocimiento de algunas características, como pueden ser los trazos en diagonal, para nosotros es claro, pero con frecuencia la identificación de tales sistemas puede resultar puramente de

cálculos arbitrarios efectuados sobre los datos de la imagen. Algunos sistemas poseen importantes usos prácticos, pero apenas si tocan la superficie de lo que entendemos por visión.

Históricamente, la visión por ordenador como comprensión de la imagen ha tenido dos enfoques opuestos: el enfoque *de arriba-abajo* o activado por modelos, y el enfoque *de abajo-arriba* o activado por datos.

El análisis de escenas activado por modelos con frecuencia se denomina *alucinación controlada.* El sistema posee un modelo interno de lo que está buscando (un autobús de dos pisos, p. ej.) y proyecta ese modelo sobre el plano de imagen en diversas orientaciones. Luego comprueba la calidad de encaje entre lo que ha «imaginado» y lo que hay verdaderamente allí.

Los sistemas de abajo-arriba exploran los datos de la imagen en busca de líneas, bordes y otros signos reveladores. Mediante esa exploración construyen una descripción simplificada de la imagen, que se denomina *boceto primitivo* y es una representación muy abstracta de los datos. La idea es que al prescindir de gran parte de los detalles se hace más fácil comparar el boceto con ejemplos tomados de una categoría dada de objetos almacenados como plantillas de referencia.

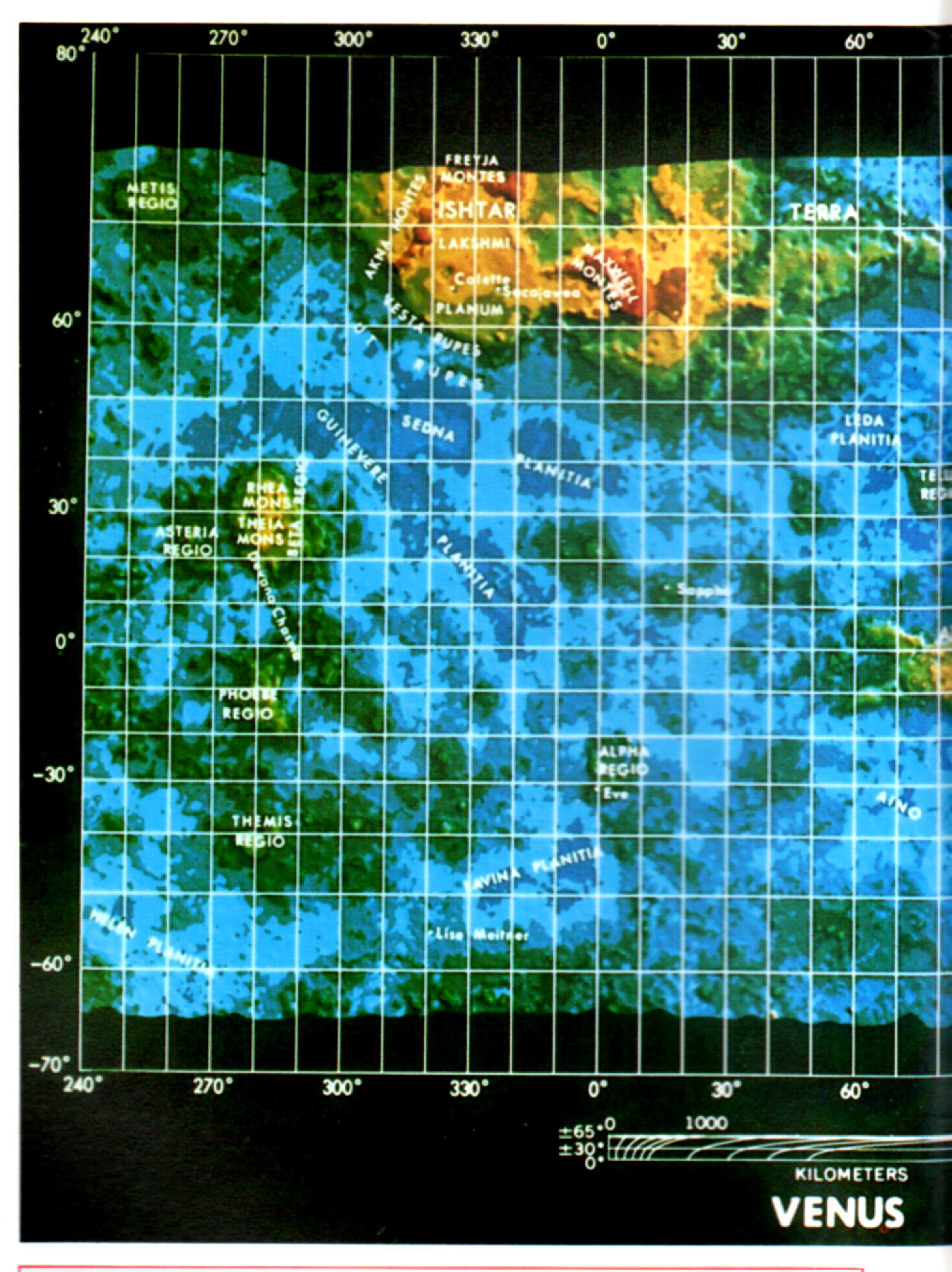
Los sistemas prácticos tienden a emplear tanto el método de arriba-abajo como el de abajo-arriba, pero hasta muy recientemente había muy pocos sistemas de aprendizaje en el campo de la visión por ordenador. Todos ellos dependían de inteligencia preprogramada.

Una excepción notable es el WISARD (Wilkie, Aleksander & Stonham's Recognition Device) de Igor Aleksander. Se le puede enseñar a hacer la distinción, comparativamente sutil, entre un rostro humano sonriente y uno con el ceño fruncido. Puede incluso generalizar este conocimiento a rostros que no haya visto nunca antes. El WISARD opera en tiempo real (25 fotogramas por segundo) y su éxito comercial para clasificar chocolates sobre una cinta transportadora móvil podría allanar el camino para una nueva generación de «máquinas que ven» que adapten y mejoren su rendimiento.

De forma muy resumida, el WISARD trabaja asignando grupos de ocho pixels por vez (óctuplos) a bancos seleccionados de RAM. Un óctuplo es un detector de rasgos y puede estar en uno de 256 estados (de 0 a 255), según el estado de cada uno de los pixels que controla. Durante la fase de entrenamiento, se almacena un 1 en la posición del banco de RAM especificada por el estado del óctuplo cuando hay presente una imagen determinada. Luego, en la fase de reconocimiento, la presencia de un 1 en la posición direccionada del banco de RAM de ese óctuplo constituye una evidencia de que la imagen «enseñada» está otra vez presente.

Mediante el empleo de una gran cantidad de óctuplos, o discriminadores, el sistema se vuelve relativamente impermeable a datos espurios (imágenes «ruidosas»).

El sistema de Aleksander posee una cuadrícula de 512 por 512 para la imagen y más de 32 000 óctuplos. Ello requiere alrededor de ocho millones de bits (un megabyte) de RAM. Tales memorias se han vuelto económicamente viables desde hace muy poco tiempo.




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

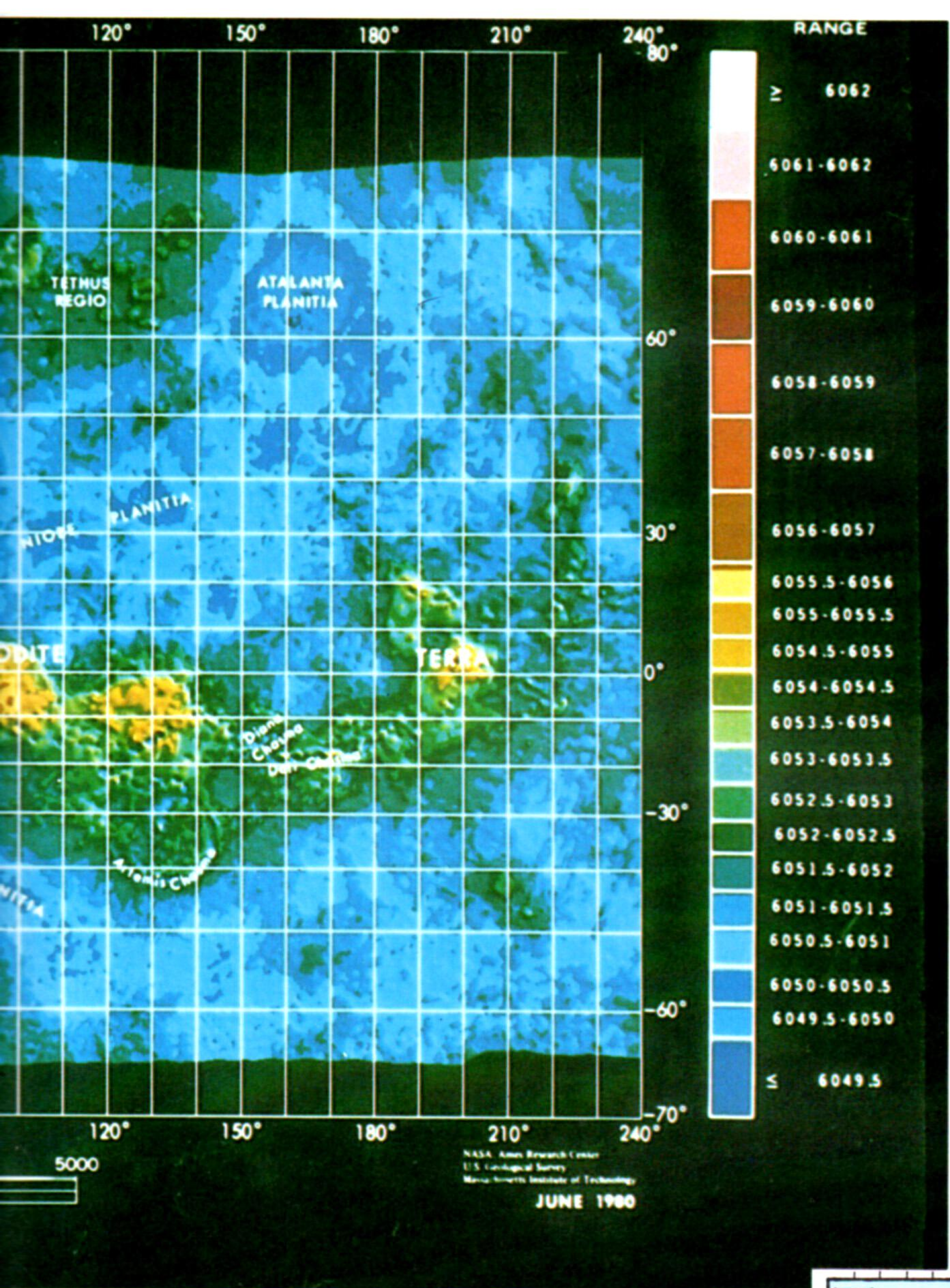
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

Caroline Clayton

## La diferencia entre R y A

En el mundo real, los datos bien definidos son muy poco frecuentes. En el ejemplo de arriba vemos formas digitalizadas de las letras A y R, pero ningún patrón está bien definido, al existir manchas y distorsiones. Los datos «ruidosos» como éstos son un problema constante para todos los tipos de sistemas de inteligencia artificial. ¿Podría usted determinar cuál es cuál?



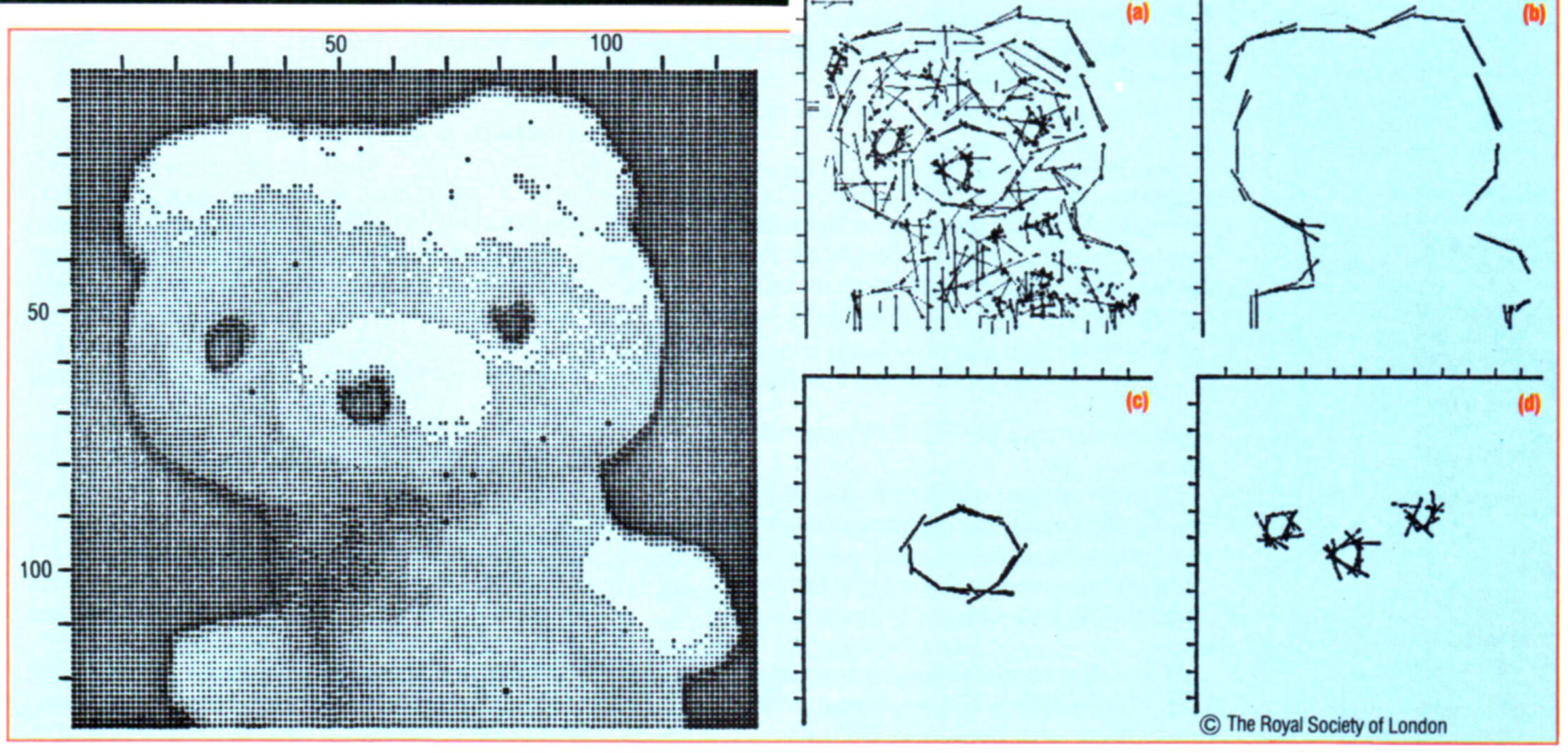



## Proceso de imágenes

Ilustramos un ejemplo de proceso de imágenes. Aquí se ha procesado por ordenador una fotografía tomada desde una sonda espacial, para acentuar las fronteras entre regiones de distinto relieve. Ello implica el uso de *colores falsos*. Lo interesante de este grabado, que en realidad es una imagen por radar del planeta Venus, es que todos los colores son «falsos». Sin la ayuda del ordenador jamás podríamos ver esta imagen, porque Venus está envuelto en densas nubes y nuestros ojos no son sensibles a las longitudes de onda que se requieren para penetrarlas. En astronomía, tales imágenes han resultado muy útiles (y algunas veces muy bellas) en los últimos años, aprovechando, además de la luz visible, las longitudes de onda de radio, infrarrojas y ultravioletas. En nuestro planeta se está investigando el lecho oceánico por sonar para construir «imágenes acústicas» que emplean el principio del sondeo acústico y no dependen en absoluto de la radiación electromagnética (luz visible, ultravioleta, infrarroja, rayos X, etc.). Otras clases de proceso de imágenes se utilizan para «limpiar» imágenes borrosas o distorsionadas. Tales algoritmos se emplean ahora de forma tan rutinaria que ya no se consideran componentes de AI

## Bocetos primitivos

Se ha desarrollado un método de proceso visual que no depende del conocimiento previo de lo que se supone que representa el patrón. El boceto primario (a) extrae de la imagen del osito de peluche fronteras y formas primitivas, mediante la comparación de regiones adyacentes. Sucesivas extracciones del boceto primitivo (de [b] a [d]) revelan grupos importantes que contribuyen a reconocer la imagen original



# Diseño funcional

**Continuando con nuestra serie dedicada al LISP, examinamos sus dos conceptos fundamentales: listas y funciones**

En el primer capítulo de esta serie vimos cómo informar al LISP de que queremos que una lista determinada se evalúe como una lista de elementos de datos y no como una función. De modo que:

```
(SETQ X '(A B C D E F))
```

asignaría la lista (A B C D E F) a la variable X, donde cada elemento de la lista es una serie de un único carácter. Veamos lo que sucedería si D, E y F fueran series de un único carácter y A, B y C fueran variables con los valores 2, 4 y 8, respectivamente. En este caso, en realidad desearíamos asignar a X la lista (2 4 8 D E F). Podemos hacerlo introduciendo una nueva función: LIST. Ésta crea una lista de sus argumentos. Por lo tanto:

```
(SETQ X (LIST 'A 'B 'C 'D 'E 'F))
```

sería exactamente lo mismo que la expresión previa, pero:

```
(SETQ X (LIST A B C 'D 'E 'F))
```

asignaría correctamente la lista (2 4 8 'D 'E 'F). Aquí, al omitir los apóstrofos precedentes, se evaluarían A, B y C. De forma muy similar:

```
(SETQ X (LIST 1 2 4 '(PLUS 4 4))
```

asignaría a X la lista (1 2 4 (PLUS 4 4)), donde el cuarto elemento de la lista es en sí mismo una lista compuesta de los tres elementos PLUS, 4 y 4. En otras palabras, PLUS no se ha evaluado porque hemos precedido la expresión con un apóstrofo. Sin embargo:

```
(SETQ X (LIST 1 2 4 (PLUS 4 4))
```

asignaría a X la lista (1 2 4 8), donde PLUS se ha evaluado (lo que demuestra la diferencia que puede suponer un simple apóstrofo). Podemos ampliar aún más este concepto para crear una lista de listas.

Por ejemplo:

```
(SETQ PERSONA' ((JUAN PI) (17 1 1960)))
```

asignaría a la variable PERSONA una lista de datos.

La lista contiene dos elementos, siendo cada uno de ellos una lista. La primera contiene dos elementos de caracteres y la segunda, tres elementos numéricos. Usted puede seguir ampliándola para obtener todavía más listas de listas.

Hasta ahora hemos visto las funciones SETQ, LIST, PLUS y TIMES. El LISP tiene muchas más funciones incorporadas, y su utilización dependerá de cada implementación en particular. Además, usted puede definir sus propias funciones; de hecho ésta es la forma en que se construyen los programas en LISP.

Antes de ver cómo se hace esto, examinemos las tres funciones básicas que existen en todas las implementaciones de LISP. Éstas son CAR, CDR y CONS. El nombre CONS alude simplemente a CONStrucción. Los nombres CAR y CDR datan de una de las implementaciones originales del lenguaje, y corresponden a las siglas de *Contents of Address Register* (contenido del registro de direcciones) y *Contents of Decrement Register* (contenido del registro de decremento), respectivamente. Tanto CAR como CDR toman un único argumento, que ha de ser una lista con al menos un elemento. Esto excluye la estructura de lista especial conocida como *lista vacía,* o NIL, que se escribe:

```
( )
```

La función CAR devuelve como resultado el primer elemento de su argumento. Así:

```
(CAR '(1 2 3 4 5))
```

devolvería el valor entero 1. No hay ningún motivo por el cual el resultado no pueda ser en sí mismo una lista.

Por ejemplo, en la expresión:

```
(CAR '((1 2) (3 4) 5))
```

el primer elemento es la lista (1 2).

La función CDR efectivamente devuelve todo menos el primer elemento de una lista. Dicho en otras palabras, devuelve lo que queda tras una operación CAR. De modo que:

```
(CDR '(1 2 3 4 5))
```

devolvería la lista (2 3 4 5), y:

```
(CDR '((1 2) (3 4) 5))
```

devolvería la lista ((3 4) 5).

CONS espera dos argumentos y los concatena entre sí, añadiendo el primer argumento a la lista del segundo argumento.

Por lo tanto:

```
(CONS 1' (2 3 4 5))
```

construiría la lista (1 2 3 4 5) y:

```
(CONS '1 2) '((3 4) 5))
```

construiría la lista ((1 2) (3 4) 5).

A menudo las funciones CAR y CDR se encontrarán anidadas, de modo que:

```
(CDR '((1 2) (3 4) 5))
```

devolverá ((3 4) 5), y:

```
(CAR (CDR '((1 2) (3 4) 5)))
```

devolverá (3 4), que es (CAR '((3 4) 5)), y:

```
(CAR (CAR (CDR '((1 2) (3 4) 5))))
```

devolverá el valor 3, que es (CAR '(3 4)).

Esto enseguida resulta tedioso, de modo que el LISP suele admitir abreviaciones. Los nombres de las funciones comienzan igualmente con C y terminan con R, pero pueden tener entremedio cualquier combinación de Aes (para CAR) y Des (para CDR). Utilizando el LISP Acornsoft, esta



combinación puede constar de hasta tres letras empotradas. Por tanto, podríamos escribir la última expresión así:

```
(CAAR (CDR '((1 2) (3 4) 5)))
```

o:

```
(CAR (CADR '((1 2) (3 4) 5)))
```

o incluso:

```
(CAADR '((1 2) (3 4) 5))
```

todas las cuales darían como respuesta 3.

## Definición de funciones

Como ya hemos observado, los programas en LISP se construyen como una serie de funciones definidas por el usuario. Hemos visto, además, que todas las expresiones son listas de funciones, de modo que es bastante natural que utilicemos funciones para definir funciones. En este caso, empleamos la función DEFUN, que espera tres argumentos de la forma:

```
(DEFUN a (b)(c))
```

donde a es el nombre de la función, b es la lista de parámetros (como las del PASCAL, el BASIC BBC, etc.) y c es una estructura de lista que contiene el cuerpo principal de la función. Estamos ahora en condiciones de definir nuestra primera función. Supongamos que frecuentemente deseáramos multiplicar números por el valor 8. Podríamos simplemente escribir:

```
(TIMES 8 N)
```

cada vez, donde N es el número a multiplicar. En cambio, vamos a definir una función para hacer lo mismo:

```
(DEFUN TIME8 (N) (TIMES 8 N))
```

Aquí la función TIME8 toma su argumento N y utiliza la función estándar TIMES para multiplicar este número por 8 y devolver un valor.

De modo que la expresión:

```
(TIME8 11)
```

ahora daría como resultado 88.

Antes de que prosigamos, hemos de examinar un importante concepto de programación: la condición. ¡Su importancia puede quedar clara si pensamos en lo que sería el BASIC sin sentencias IF...THEN!

La sentencia condicional del LISP asume la forma de una función:

```
(COND (Condición1 Expresión1)
      (Condición2 Expresión2)
           :          :
      (CondiciónX ExpresiónX))
```

En primer lugar, observe que esta función se ha extendido a varias líneas para darles cabida a todas.

Esto es bastante normal en LISP, que sabe cuándo usted ha terminado una expresión porque habrá cerrado el paréntesis final.

La función COND es muy parecida a la sentencia CASE del PASCAL. Se evalúa la Condición1 y, de ser verdadera, entonces se utilizará la Expresión1 para devolver un resultado. De lo contrario, se comprobará la Condición2, y así sucesivamente. Algunas implementaciones de LISP (incluyendo el LISP Acornsoft) exigen que la evaluación de al menos una de las condiciones (denominadas *predicados*) resulte verdadera. Esto se maneja fácilmente mediante la adición de una condición final de la forma:

```
(COND (Condición1 Expresión1)
      (Condición2 Expresión2)
           :          :
      (T          Expresión final))
```

Aquí el carácter T se utiliza para representar T*rue*; de modo que la expresión final (en caso de que se llegue a ella) siempre se evaluará. En LISP:

```
T = True = No cero
```

y:

```
F = False = Cero = (  )
```

Lo último es la lista vacía. Ahora podemos definir una función algo más compleja. Una facilidad muy útil de la que disponen la mayor parte de las versiones de BASIC es ABS, que devuelve el valor absoluto de su argumento. En otras palabras, si su argumento es positivo, quedará sin modificar; de lo contrario, será negado.

Nuestra función de LISP habrá de ser de la forma:

```
(DEFUN ABS (X)
   (cuerpo de la función))
```

donde X es el argumento entero. Empleando nuestra nueva función condicional, podemos escribir toda la función ABS como:

```
(DEFUN ABS (X)
   (COND ((MINUSP X) (MINUS X))
     (T             X            )))
```

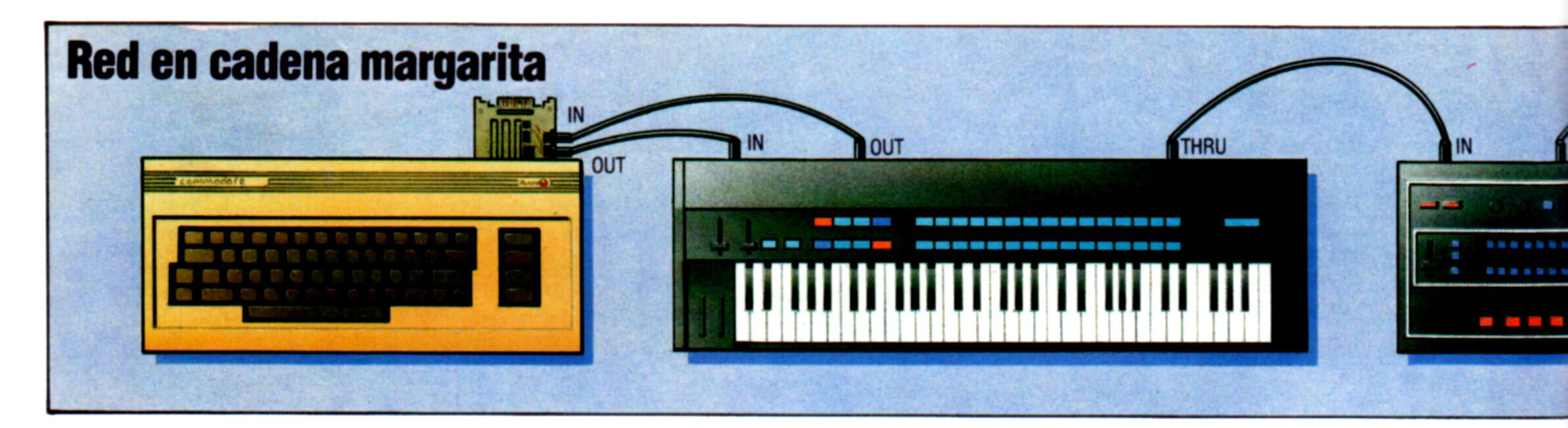
Aquí hemos utilizado dos nuevas funciones. MINUSP es una función de comprobación que devuelve el valor si su argumento es un número negativo, y falso de lo contrario. Si la condición evalúa a T, la función MINUS cambia su argumento a negativo. Si no es éste el caso, la condición T (que siempre es verdadera) devolverá el valor X original.

```
(DEFUN EQUAL (A B) (COND
  ((EQ A B) T)
  ((OR (ATOM A) (ATOM B)) NIL)
  ((EQUAL (COCHE A) (COCHE B)) (EQUAL (CDR A) (CDR B)))
  (T NIL)))
(DEFUN HALLAR_COCHE (X L) (COND
  ((NULL L) '(NINGUN NUMERO))
  ((EQUAL X (CDAR L)) (COCHE L))
  (T (HALLAR_COCHE X (CDR L)))))
(SETQ COCHES '((PEREZ ABC123X) (TORRES XYZ789Y)
          (MARTIN ACE99L) (LUCIA JBB1))
(PRINTC (HALLAR_COCHE '(ACE99L) COCHES))
```

**Procedimiento de búsqueda**
Este sencillo programa de recuperación de datos demuestra el uso de las funciones CDR y EQ. El número de la matrícula de un automóvil, entrado por el usuario, devolverá, de estar incluido en la base de datos del programa, una lista de dos elementos: el número de matrícula y el apellido del propietario del coche

**Funciones de comprobación**
He aquí una lista de otras funciones de comprobación que se suelen utilizar en LISP:

| | |
|---|---|
| (OR arg 1 arg 2...) | Devuelve True si al menos uno de sus args. es verd. |
| (ONEP arg) | Devuelve True si arg=1 |
| (ZEROP arg) | Devuelve True si arg=0 |
| (LISTP arg) | Devuelve True si arg es una lista y False si es un átomo |
| (ATOM arg) | Devuelve True si arg es un átomo y False si es una lista |
| (LESSP arg1 arg2) | Devuelve True si arg1 < arg 2 |
| (GREATERP arg1 arg2) | Devuelve True si arg1 > arg 2 |
| (EQ arg1 arg2) | Devuelve True si arg1=arg2 |




**Margarita, Margarita...**
El método más usual para conectar instrumentos compatibles con la MIDI es mediante una cadena margarita. La mayoría de los instrumentos poseen conectores MIDI THRU (además de MIDI IN y MIDI OUT). Éstos, en efecto, permiten la formación de un sistema tipo bus, en el cual todos los instrumentos están conectados al bus pero sólo responden a mensajes por canales determinados

# Reglas de composición

## Construida ya la placa de la interface, concentraremos nuestra atención en el software

Ya hemos visto cómo se transmite un byte de datos a través de la MIDI hasta un instrumento receptor. Ahora vamos a determinar el formato del (los) byte(s) necesarios para comunicar la información requerida. Un único «evento» musical se transmite como un grupo de bytes denominado *mensaje*. La longitud de la mayoría de los mensajes está entre uno y tres bytes, con la excepción de los mensajes «exclusivos para el sistema» (que analizaremos más adelante), cuya longitud puede ser de cualquier número de bytes. Cada mensaje comienza con un byte cuyo bit más significativo (MSB) es igual a uno, seguido por el resto de los bytes del mensaje, todos los cuales tienen sus MSB establecidos en cero. Se dice que un byte con su MSB establecido en uno es un byte de *estado*; los otros son bytes de *datos*. Los mensajes MIDI se dividen en dos tipos básicos: *canales* y *sistemas*.

La necesidad de los *mensajes de canal* surge del «encadenamiento en margarita», el método de interconexión más usual utilizado en sistemas pequeños. En estos sistemas, cada unidad recibe todos los datos enviados por la unidad transmisora maestra. Los mensajes de canal se utilizan para transmitir información que no esté dirigida necesariamente a todas las unidades de un sistema. Por consiguiente, a la mayoría de los receptores MIDI se les puede asignar un *número de canal* entre 1 y 16.

Los mensajes de canal poseen bytes de estado con valores comprendidos en la escala entre $80 y $EF inclusive, y tienen uno o dos bytes de datos. El número de canal está codificado en los cuatro bits menos significativos (el dígito hexa menos significativo) del byte de estado, representando $0 el canal 1 y $F el canal 16. Los tres bits restantes determinan el tipo de mensaje que viene a continuación.

Una característica especial de los mensajes de canal es que no es necesario enviar el byte de estado si se trata del mismo que el byte de estado anterior. En efecto, el estado actual o *corriente* permanece vigente hasta que se recibe otro byte de estado. La excepción es cuando un mensaje de sistema en tiempo real interrumpe temporalmente el estado actual.

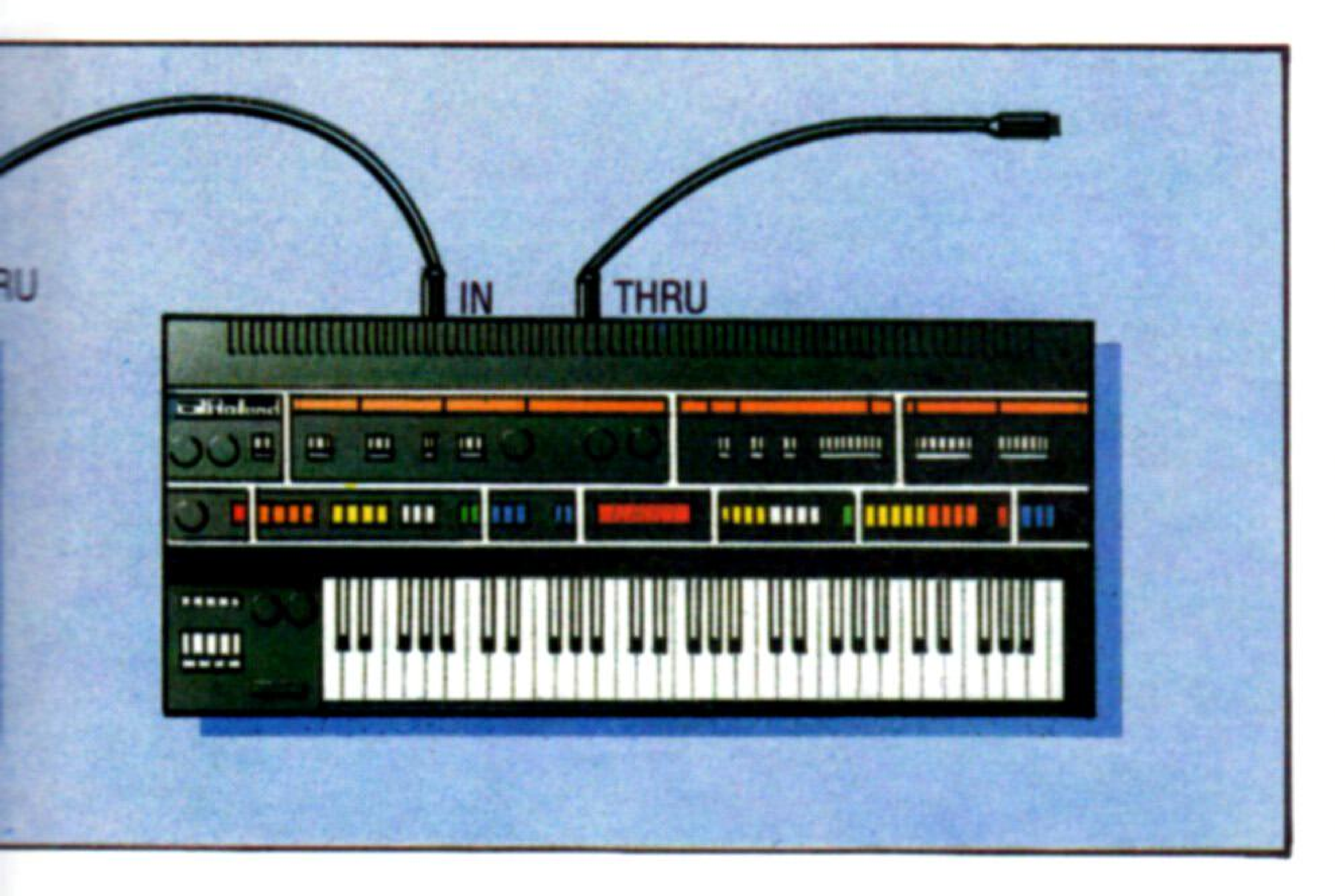
Esto es particularmente útil para comunicar muchos mensajes consecutivos de nota *on/off*, dado que un mensaje *nota on* con una velocidad (con cuánta fuerza se pulsa una tecla) de cero equivale a un mensaje *nota off*. La utilización del mismo estado (*nota on*) para una serie de mensajes de nota *on/off* resulta en un ahorro de espacio de memoria y de tiempo de transmisión.

En los sintetizadores, *asignación de voz* es el proceso de dirigir un mensaje de nota (ya sea desde la MIDI o desde el teclado del instrumento) a una de las voces del sintetizador disponibles en ese momento para producir realmente la nota. Por ejemplo, se dice que un sintetizador polifónico de seis notas posee seis voces. Para controlar la respuesta del instrumento a los mensajes de canal, los *mensajes de modalidad* MIDI pueden seleccionar cierto número de modalidades. El receptor que no sea capaz de operar en la modalidad requerida ignorará el mensaje de modalidad.

La *modalidad omni* es la más simple; en ella el receptor responde a todos los mensajes de canal con independencia del número de canal MIDI codificado en los mensajes. Cuando se apaga la modalidad omni, la unidad responde sólo a los mensajes enviados por canales MIDI específicos.

La mayoría de los sintetizadores polifónicos poseen un número de voces limitado (por lo general, seis u ocho), lo que significa que debe emprenderse alguna acción si se requiere una nota cuando ya han sido asignadas todas las voces por mensajes de nota anteriores. Por lo general las voces se asignan por estricto orden de rotación, de modo que la nota menos reciente se apaga para que la voz quede disponible para la nueva nota. Esta modalidad de operación se selecciona mediante el mensaje *poly*






*mode on*. Un mensaje *mono mode on* indica al receptor que asigne cada una de sus voces monofónicamente a uno de un grupo de canales MIDI consecutivos comenzando desde el canal original (básico) del sintetizador. El mensaje de modalidad mono se envía por el canal básico y su segundo byte de datos especifica el número total de canales requeridos.

Si está activada la modalidad omni, un mensaje mono simplemente indica al receptor que asigne monofónicamente una voz a los mensajes de canal MIDI en todos los canales.

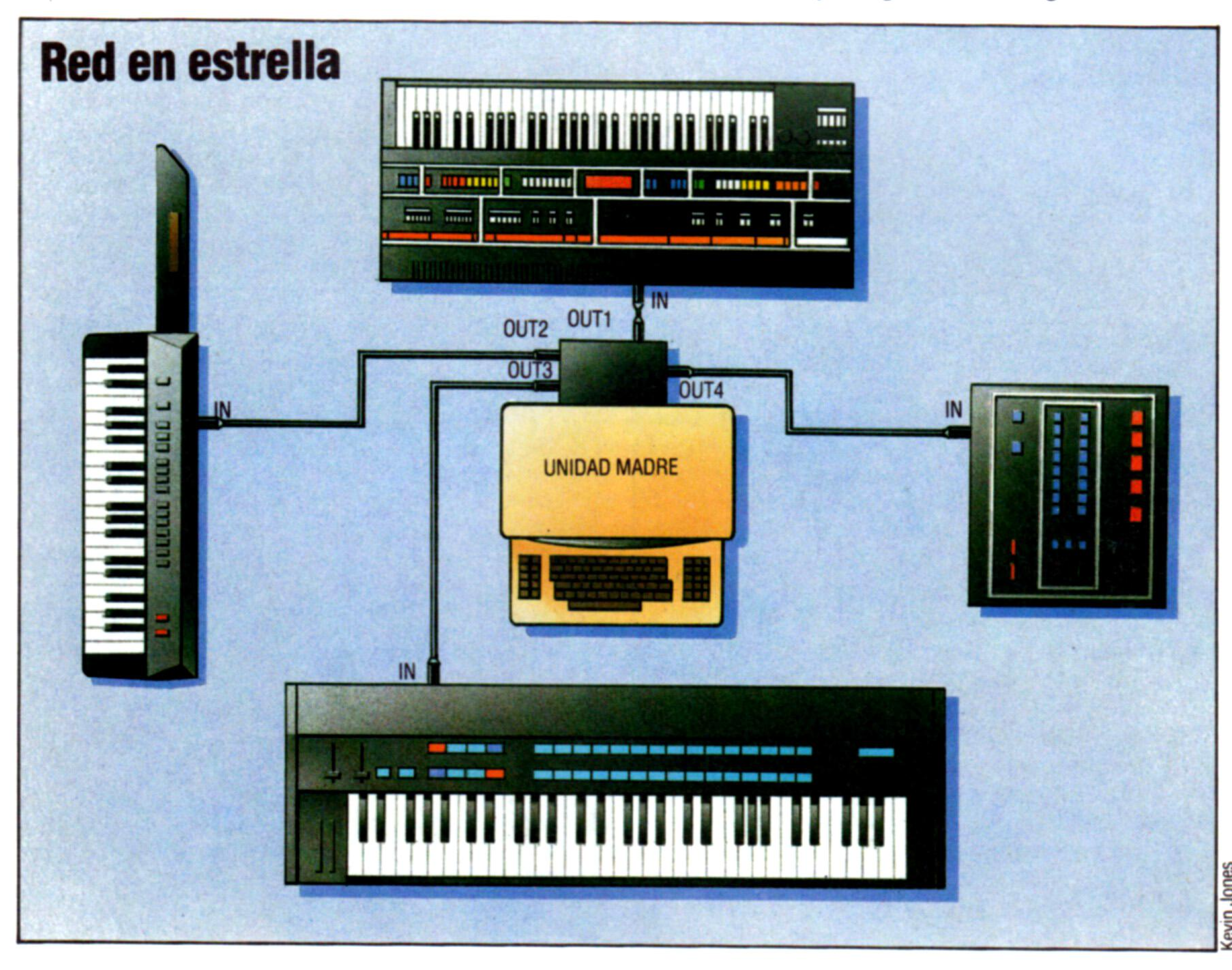
Los *mensajes del sistema* no se codifican con números de canal en sus bytes de estado. Por consiguiente, todas las unidades del sistema reciben el byte de estado. Los mensajes del sistema caen en tres categorías: comunes, en tiempo real y exclusivos.

Los mensajes de *sistema comunes* poseen bytes de estado desde $F1 hasta $F7. Se componen por el byte de estado seguido de 0, 1 o 2 bytes de datos.

Los mensajes del *sistema en tiempo real* están dirigidos a todas las unidades del sistema. Tienen bytes de estado desde $F8 hasta $FF y los utilizan principalmente las máquinas de ritmos y los secuenciadores para sincronizar sus propias secuencias internas con el reloj del transmisor maestro. La mayoría de los sintetizadores ignorarán estos mensajes a menos que posean alguna forma de secuenciador interno con capacidad para sincronización MIDI.

Los mensajes del sistema en tiempo real difieren de los otros mensajes en que se componen de un byte de estado solamente y ningún byte de datos. En consecuencia, se permite enviarlos en cualquier momento, aun cuando interrumpan mensajes de otros tipos.

Los mensajes *exclusivos del sistema* comienzan con el byte de estado $F0, seguido por cualquier número de bytes de datos, y acaban ya sea con el byte de estado de final de exclusivo ($F7) o bien con cualquier otro byte de estado. El primer byte de datos es el código de identificación (ID) del fabricante. Si éste no coincide con el ID de la unidad receptora, se ignorará el resto del mensaje. Los mensajes están destinados a la transferencia de datos entre instrumentos de tipo similar, que carecerían de significado para otras unidades. El principal tipo de datos transferido son los datos de *programa parche* para sintetizadores. (No debe confundirse la transmisión de un programa parche de esta forma con el mensaje de canal $Cx, que selecciona entre programas parche ya almacenados en la memoria del receptor.) Sin embargo, el tipo de datos transmitido por los mensajes exclusivos del sistema queda en manos del fabricante, siempre y cuando se le haya asignado un código ID válido.




**Una actuación estelar**
Las primeras implementaciones de la MIDI no permitían la selección de canales, de modo que todos los sintetizadores intentaban responder a todos los canales, con lo cual el sistema resultaba casi inútil cuando tales dispositivos estaban encadenados en margarita. Estas primeras unidades MIDI habían de estar conectadas en una formación de estrella alrededor de una unidad madre, que incorporaba un sistema de circuitos de modo tal que pudiera obtenerse cada canal desde un conector OUT separado

# Mensajes de canal de voz

| Estado | Data1 | Data2 | Descripción | Notas |
|---|---|---|---|---|
| $8x | p | v | Nota apagada<br>p=altura<br>v=velocidad fuera | 1,2 |
| $9x | p | v | a)v>0<br>Nota encendida<br>p=altura<br>v=velocidad | 1,2 |
| | | | b)v=0<br>Nota apagada<br>p=altura | 1,2 |
| $Ax | p | pr | Polifónico después de pulso<br>p=altura<br>pr=presión | 1,3a |
| $Bx | n | c | a)n<$7A<br>Cambio de control<br>n=número controlador<br>c=valor controlador | 4 |
| | | | b)n=$7A<br>Control local<br>on(c=$7F)<br>off(c=0) | 5 |
| | | | c)n=$7B,c=0<br>Todas las notas apagadas | 6 |
| | | | d)n=$7C,c=0<br>Mod. omni apagada<br>(todas las notas apagadas) | 6 |
| | | | e)n=$7D,c=0<br>Mod. omni encendida<br>(todas las notas apagadas) | 6 |
| | | | f)n=$7E<br>Mod. mono encendida<br>(todas las notas apagadas)<br>c= número de canales | 6,7 |
| | | | g)n=$7F,c=0<br>Mod. poli encendida<br>(todas las notas apagadas) | 6 |
| $Cx | pp | — | Selección Prog. Parche<br>pp=número de programa | |
| $Dx | pr | — | Presión de canal<br>pr=valor de presión | 3b |
| $Ex | 1 | m | Cambio mando de altura<br>1=7 LSBs<br>m=7 MSBs | 8 |

**Notas:**
Excepto donde se especifique otra cosa, todos los bytes de datos pueden tener cualquier valor entre 0 y $7F. El número de canal se representa mediante x, el dígito (hexa) menos significativo del byte de estado (x=0 especifica canal 1 y x=$F especifica canal 16).
1. p es el tono de la nota en semitonos. *Do central* es el número de tono $3C y, por tanto, todos los *do* son múltiplos de $0C (decimal 12). El teclado de piano estándar de 88 notas va de $15 a $6C.
2. Los valores de velocidad van desde $01 hasta $7F. Los teclados sin sensores de velocidad deben utilizar una velocidad por defecto de $40. Un mensaje de «nota encendida» con una velocidad de 0 equivale a un mensaje de «nota apagada» con una velocidad por defecto de $40.
3. «Después de pulsación» es la cantidad de presión ejercida sobre una tecla tras haberla pulsado. Por lo general se utiliza para introducir efectos de modulación sin tener que operar un mando de modulación. Hay dos tipos de sensores de presión y dos tipos diferentes de mensajes.
a) El "después de pulsación" individual o polifónico requiere sensores de presión individuales para cada tecla y afecta sólo a la nota correspondiente a la tecla pulsada. En consecuencia, además del valor de presión se debe enviar un número de tono. (Su implementación requiere, asimismo, que se proporcionen circuitos de modulación separados para cada voz.)
b) La presión de canal se produce mediante un único sensor de presión, al que afectan de igual forma todas las teclas. La presión que se transmite corresponde a la presión máxima (todas las teclas pulsadas al mismo tiempo). El efecto resultante se aplica simultáneamente a todas las voces.
4. Estos mensajes los envían controladores separados del teclado. Los números controladores de $0 a $1F son controladores continuos que poseen valores de control (c) de $0 a $7F. Éstos equivalen a las palancas de mando de potenciómetro para ordenadores y de hecho pueden ser palancas de mando u otros dispositivos tales como mandos de ruedas, pedales y controladores de aliento. Los números de controlador de $20 a $3F se utilizan opcionalmente para enviar siete bits menos significativos extras para los controladores de $0 a $1F, si se requiere una resolución muy alta. Los números de controlador del $40 al $5F son controladores de conmutación (como los pedales de sostenimiento, portamento, interruptores on/off, etc.) con c establecido ya sea en cero (off) o bien en $7F (on). Éstos son equivalentes al tipo de palanca de mando de conmutación más usual que utilizan los ordenadores. Los números de controlador del $60 al $79 son indefinidos, y los del $7A al $7F están reservados para mensajes de modalidad de canal. No se requiere que se asignen los números de controlador a controladores físicos específicos, con la excepción del mando de modulación, al que se suele asignar el $1.
5. El control local se utiliza opcionalmente para romper la unión interna entre los dispositivos de entrada de una unidad (por lo general, un teclado y controladores asociados) y su sistema de circuitos de generación de sonido, de modo que el teclado (p. ej.) sólo envía datos a MIDI OUT y los circuitos de generación de sonido sólo responden a los datos recibidos en MIDI IN.
6. La implementación de «todas las notas apagadas» es opcional, y estos mensajes no se deben utilizar en lugar de instrucciones individuales de «nota apagada» para apagar varias notas. El razonamiento que motiva esto no está nada claro y hace que las instrucciones sean virtualmente inútiles.
7. El tercer byte del mensaje de modalidad mono especifica el número total de canales requerido. Si este número es cero, la cantidad de canales es igual a la cantidad de voces disponibles en el receptor.
8. El mando de altura se diferencia de otros controladores en que puede tener valores positivos y negativos. La posición central se envía como l=$0 y m=$40. Muchos receptores poseen sólo siete bits de resolución y sólo responderán a m, ignorando el valor de l

# Mensajes del sistema

| Estado | Datos1 | Datos2 | Descripción | Notas |
|---|---|---|---|---|
| $F0 | cualquier número | | Exclusivo del sistema | 1 |
| $F1 | | | Indefinido | |
| $F2 | l | m | Señalador posición canción<br>l=7 LSBs<br>m=7 MSBs | 2 |
| $F3 | s | | Selección de canción<br>s=número de canción | |
| $F4 | | | Indefinido | |
| $F5 | | | Indefinido | |
| $F6 | | | Solicitud de melodía | 3 |
| $F7 | | | Fin de exclusivo del sistema | 1 |
| $F8 | — | — | Reloj temporizador | 4 |
| $F9 | | | Indefinido | |
| $FA | — | — | Empezar | 4 |
| $FB | — | — | Continuar | 4 |
| $FC | — | — | Parar | 4 |
| $FD | | | Indefinido | |
| $FE | — | — | Percepción activa | 5 |
| $FF | — | — | Inicialización del sistema | 6 |

**Notas:**
Los mensajes de $F8 a $FF son los mensajes del sistema en tiempo real y se pueden enviar en cualquier momento (incluso durante la transmisión de otros mensajes).

1. Los mensajes exclusivos del sistema pueden tener cualquier número de bytes de datos, terminando con un byte de «fin del exclusivo del sistema» ($F7) o cualquier otro byte de estado.

2. Este mensaje se utiliza para preestablecer arbitrariamente el puntero de posición de canción, que es un registro interno que contiene la cantidad de *beats* (1 *beat*=6 relojes MIDI) desde el comienzo de la secuencia (canción).

3. Éste se utiliza para solicitar a los sintetizadores analógicos que afinen sus osciladores.

4. Estos mensajes se emplean para sincronizar las unidades secuenciadoras maestra y esclava. El reloj del sistema se establece a una velocidad de 1/24avo de un cuarto de nota. La instrucción «continuar» se diferencia de «comenzar» en que reinicia una secuencia desde el puntero en el cual se detuvo.

5. La percepción activa se utiliza como un mensaje ficticio cuando en la MIDI no se registra ninguna otra actividad. Si se utiliza, debe ser enviado de modo que no transcurran más de 300 ms sin actividad.

6. Este mensaje se emplea para inicializar el sistema completo a la situación de encendido. No se lo debe enviar automáticamente detrás del encendido, para impedir la posibilidad de que dos unidades se reinicialicen la una a la otra indefinidamente.

No se deben enviar mensajes que no estén incluidos en la lista anterior, y los receptores habrán de ignorarlos



# Pequeño melocotón

## Si bien el Apricot F1e es principalmente una máquina de gestión, puede también suscitar interés en el campo educativo

Quienes estén familiarizados con los ordenadores Apricot (albaricoque) reconocerán fácilmente la estructura básica del F1e. Vendido en forma de paquete, incluye el teclado F1, la unidad del ordenador propiamente dicha con una unidad de disco de una sola cara y densidad simple, y una pantalla de fósforo verde de ocho pulgadas. Al igual que con el Apricot Portable, la empresa ha aplicado su tecnología de infrarrojos, que elimina la necesidad de usar gran número de cables colgantes. Tanto el teclado como el ratón opcional controlan el ordenador a través de haces infrarrojos que son detectados por sensores en la parte delantera del F1e. Haciéndose eco de la tendencia hacia ordenadores cada vez más pequeños (iniciada con el Apple Macintosh), que no ocupan excesivo espacio, el F1e es notablemente estrecho, midiendo apenas 200 mm de ancho, aunque esto queda compensado por su longitud, que es de alrededor de 425 mm.

El teclado tiene el trazado Apricot estándar; las teclas están niveladas y su aspecto es similar a las del Sinclair QL. No es particularmente apropiado para mecanografía al tacto y, en consecuencia, no se presta muy bien a aplicaciones de tratamiento de textos. Las teclas, si bien son adecuadas para una máquina de gestión moderna, traquetean un poco y pueden despertar dudas en cuanto a su fiabilidad a largo plazo. Al igual que en el Apricot Portable, en la parte superior del teclado hay cuatro botones: «Reset», para arranques en frío, «Repeat Rate», que permite variar la velocidad a la cual se repiten los caracteres cuando se mantiene pulsada una tecla, «Set Time» y «Keyboard Lock».

La unidad del ordenador posee una única unidad de disco Sony de 3 ½ pulgadas, incorporada en la parte anterior, que está adquiriendo una creciente popularidad entre los fabricantes y en consecuencia ha justificado el uso original de ACT de este formato para sus máquinas en más de un sentido. Mientras que otras empresas están teniendo dificultades para transferir sus amplias bases de software a los discos de 3 ½ pulgadas, todo el software Apricot es convenientemente compatible. En consecuencia, los usuarios que deseen adquirir el F1e no tienen necesidad de preocuparse por la falta de software adecuado. Además, los discos de una sola cara y densidad simple que utiliza el F1e pueden retener un máximo de 315 K de información, más que sus equivalentes de 5 ¼ pulgadas.

Otras características de la parte delantera de la unidad de ordenador incluyen una serie de LEDs que indican potencia, Caps Lock, desplazamiento de la pantalla y unidad de disco *on/off*. Debajo de ellos están los detectores de haces infrarrojos.

En el lado derecho del ordenador hay un bus de ampliación de 60 vías, a través del cual se puede añadir una amplia gama de placas de ampliación o una unidad de disco extra. Esta ranura hace que el F1e se pueda ampliar a la misma capacidad que el Apricot XL mediante la adición del sistema de ampliación MSD, que proporciona 10 megabytes en disco rígido.

El panel posterior, si bien no posee muchas interfaces, está diseñado suficientemente bien como para permitir la instalación de los periféricos más comunes. Sobre la izquierda hay un conector D de 25 vías estándar que da cabida a una interface en serie RS232. Junto a esta puerta hay dos conectores para pantalla. El primero es un adaptador de nueve patillas que proporciona la señal RGB para pantallas en color (si bien las pantallas ACT monocromáticas existentes también se enchufan en este conector). A la derecha de éste hay un conector de video compuesto para otros tipos de pantallas.

El precio del F1e no incluye la pantalla. El ordenador no posee ningún adaptador RF que lo capacite para enviar una señal a un aparato de televisión común; no obstante, sí hay disponible un adaptador




Chris Stevens

**Aspecto atrayente**
El Apricot F1e ha heredado la apariencia atractiva de sus parientes. La unidad de disco integral de 3 ½ pulgadas, el sistema operativo MS-DOS estándar y los paquetes de software en lote, junto con un precio asequible, hacen del F1e un serio competidor en los círculos educativos y también entre los usuarios serios de ordenadores personales

que proporciona tal señal. Asimismo, dado que el F1e estándar no tiene pantalla instalada y que muchas pantallas poseen su propia entrada de potencia, los conectores para pantalla no proporcionan líneas de potencia de forma automática, y el ordenador propiamente dicho no es capaz de proporcionar tal fuente. Por lo tanto, para que éste pueda operar una de las pantallas ACT, se le ha de instalar una fuente de alimentación externa de 17 V.

Aunque esto parezca lógico, el hecho de que ACT haya tomado la decisión de suprimir los cables colgantes mediante la incorporación de comunicaciones infrarrojas, no concuerda con su decisión de proporcionar una fuente de alimentación externa para operar sus pantallas. Esto adquiere especial relevancia teniendo en cuenta que otros modelos de la gama (como el ACT Apricot, el buque insignia de la empresa) no requieren tal dispositivo. La fuente de alimentación se puede colocar en un lugar donde no moleste, en especial cuando se la instale en una posición permanente, pero así y todo uno queda con la sensación de que, en este aspecto, la máquina no se pensó lo suficiente.

En el interior del ordenador, hay otra ranura de ampliación para placas adicionales. La placa del F1e, al igual que otras de la gama Apricot, tiene un diseño elegante y está protegida de la unidad de disco y el transformador de potencia (que puede generar temperaturas perjudiciales) mediante una carcasa de metal que actúa a modo de disipador.

El F1e está basado en el procesador 8086 de 16 bits y opera bajo el popular sistema operativo MS-DOS. Los ordenadores Apricot, sin embargo, siempre han tenido su propio sistema de administración basado en iconos, lo que en realidad significa que su relación directa con el sistema operativo es escasa. El sistema de administración del F1e se llama Activity y es con él con quien los usuarios se familiarizarán más.

Activity es un programa orientado hacia objetos. Ello significa que, en lugar de impartir una serie de instrucciones para el ordenador, como una para cargar (LOAD) un archivo en la memoria, uno sola-




**Software de calidad**
El software gratuito que se entrega con el F1e incluye un procesador de textos, un cuaderno de notas electrónico y una hoja electrónica. Aunque los dos primeros programas son, relativamente, poco sofisticados, la hoja electrónica (*SuperCalc*) posee muchas facilidades, como la división en ventanas, que cabe esperar de todas las aplicaciones caras de este tipo

**Chip Z80 SIO**
Proporciona las señales de reloj para sincronizar las acciones del chip de E/S en serie

**ROMs F1**
Estos chips de ROM incluyen las rutinas cargadoras, de autodiagnóstico y BIOS para el sistema

**Chips RAM**
El F1e incluye 256 K de RAM

**Controlador de disco flexible**
Este chip lógico hecho a medida controla las acciones de la unidad de disco

**Unidad de potencia**
La unidad de potencia incluye un transformador de corriente para el ordenador y un conector CA de 17 V a través del cual se debe alimentar la fuente de potencia para la pantalla

**Luces indicadoras**
Informan de las condiciones de potencia conectada, caps lock, desplazamiento de pantalla y acceso al disco

**Receptor infrarrojo**
El teclado sin cables del F1e utiliza transmisión por infrarrojos para comunicarse con la unidad principal




Chris Stevens

mente indica (por lo general seleccionando un icono entre una serie presentada en la pantalla) la acción que desea realizar: el ordenador se encargará del resto.

La mayoría de estos sistemas dependen del uso de un ratón para desplazar el cursor a través de la pantalla; pero en el sistema F1e el mismo se puede sustituir por el teclado numérico. Los números del 1 al 9 están dispuestos en tres filas de tres teclas. Tomando el teclado a modo de brújula (excluyendo la tecla central) hay ocho direcciones en las cuales se puede mover el cursor. Por lo tanto, pulsando la tecla central de la fila superior (8) el cursor se moverá directamente hacia arriba hasta un icono que se halle en esa dirección, mientras que al pulsar 3 (tecla de la derecha de la fila inferior) el cursor se moverá oblicuamente hacia abajo y hacia la derecha. Para seleccionar un icono se debe pulsar la tecla Enter del teclado de calculadora.

Entre el juego de programas incluido en el disco del sistema Activity hay un programa de aprendizaje para ayudar al usuario a acostumbrarse al sistema. Asimismo, explica los usos del icono y los editores de fuentes (con facilidades para cargar sus propios juegos de caracteres cuando así lo quiera) y el configurador del sistema, que permite preparar el ordenador para cualquier periférico especializado que pueda estar utilizando.

Como cabe esperar en una máquina pensada fundamentalmente como máquina de gestión, el F1e incluye varios paquetes de software en un lote de aplicaciones. *SuperWriter* es un programa para tratamiento de textos basado en el *WordStar*, cuyo uso está tan difundido, pero sin las amplias capacidades de formateo de este último. El *SuperPlanner* se describe como un «bloc de notas electrónico» que permite al usuario «planificar con antelación» sus actividades. Este programa incluye una agenda de direcciones, un calendario, un diario y un pequeño sistema de archivo. Aunque superficialmente se asemeja a una base de datos, *SuperPlanner* no contiene las refinadas técnicas de búsqueda y recuperación que por lo general caracterizan a una auténtica base de datos.

El último paquete que viene empaquetado con el F1e es *SuperCalc*. Éste es una hoja electrónica para aplicaciones financieras y contables. Quizá sea el más completo de los tres paquetes; contiene una gran cantidad de instrucciones que permiten ventanas, justificación de textos y muchas otras facilidades que es dable esperar de un paquete de hoja electrónica profesional.

Al reducir el precio del F1e, ACT ha dado a entender que tiene intenciones de introducirse de forma concertada en el mercado educativo. Esta actitud se hace aún más evidente al haber lanzado ACT, paralelamente al anunciado recorte de precios, un nuevo producto denominado B-TRAN, que permite que el ordenador ejecute virtualmente todos los programas escritos en BASIC BBC.

Ciertamente, la reducción del precio del F1e mejorará las ventas del ordenador, en particular en los mercados educativo y de pequeña gestión. Si bien no es probable que llegue a dominar el mercado del ordenador personal, el F1e es, no obstante, una compra excelente para quienes estén interesados en adquirir una máquina de precio asequible que se puede ampliar hasta alcanzar la potencia de un ordenador de gestión totalmente equipado.

## APRICOT F1e

**DIMENSIONES**

425 × 200 × 105 mm

**CPU**

Procesador Intel 8086 operando a 4.7 MHz

**MEMORIA**

256 Kbytes estándares

**PANTALLA**

Resolución de textos de 132 × 50 u 80 × 25 caracteres, o una resolución máxima para gráficos de 800 × 400 pixels

**INTERFACES**

Puerta RS232, interface Centronics y conectores para RGB y video compuesto

**UNIDAD DE DISCO**

Una sola unidad de disco de 3 ½ pulgadas y 315 Kbytes

**SISTEMA OPERATIVO**

MS-DOS, CP/M-86 y Concurrent CP/M

**DOCUMENTACION**

El manual del paquete de aplicaciones es muy completo y responde al elevado estándar que es habitual en ACT, si bien el manual de iniciación es algo parco en cuanto a información detallada sobre la máquina en sí misma

**VENTAJAS**

Por su precio, el F1e parece una ganga y podría causar un gran impacto en los mercados educativo y de pequeña gestión

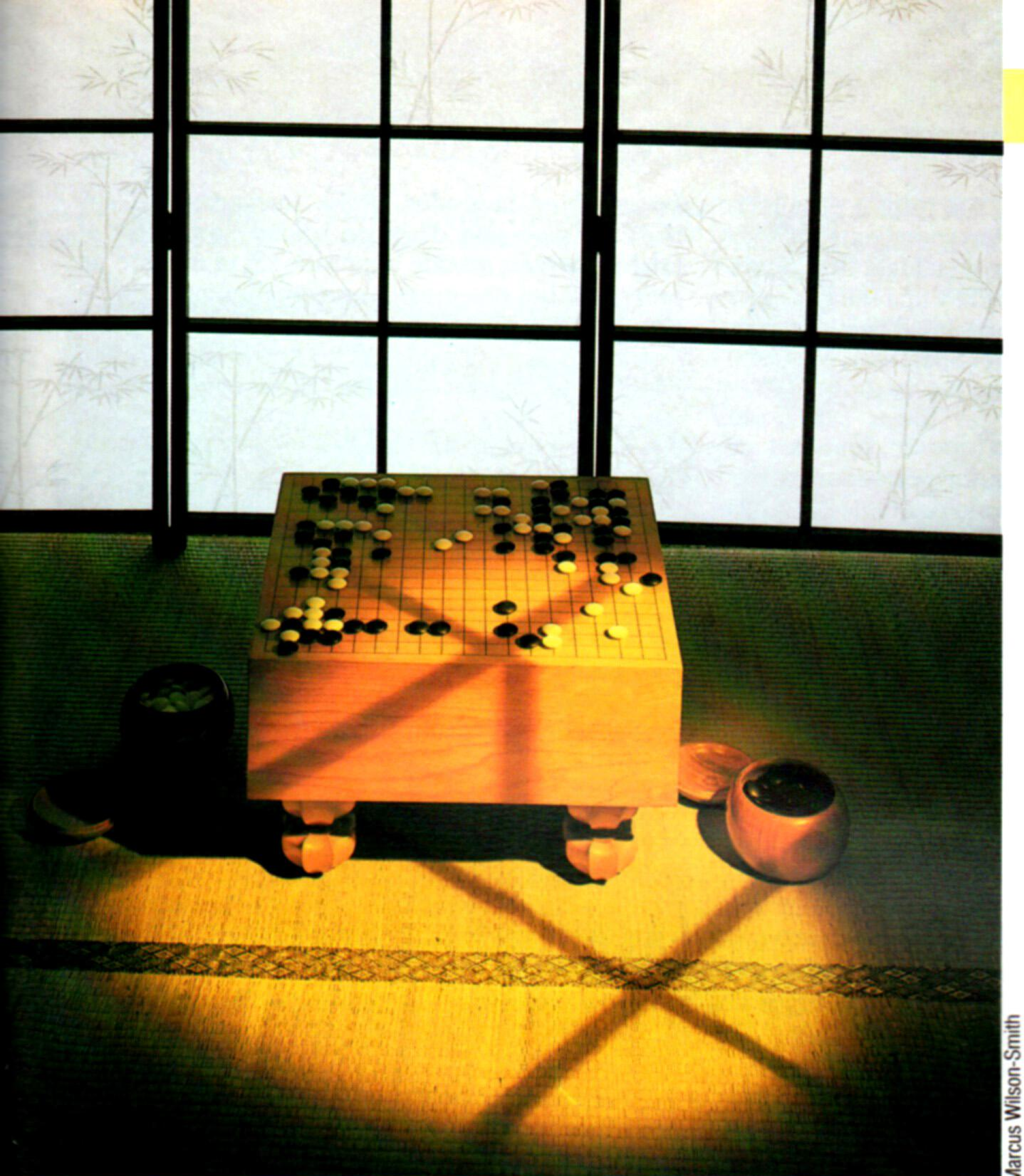
**DESVENTAJAS**

El teclado plano no está a la altura del estándar del ACT Apricot. Aunque contiene un procesador rápido, el F1e no es tan veloz como algunas máquinas que pueden compararse con él

# Datos básicos (I)

**Por cortesía de la Commodore Business Machines, iniciamos un análisis detallado del mapa de memoria del Commodore 64**

| ETIQUETA | DIRECCIÓN HEXA | POSICIÓN DECIMAL | DESCRIPCIÓN |
|---|---|---|---|
| D6510 | 0000 | 0 | Chip 6510 On-Registro de dirección de datos |
| R6510 | 0001 | 1 | Chip 6510 On-reg. E/S 8 bits |
| | 0002 | 2 | No usado |
| ADRAY1 | 0003-0004 | 3-4 | Vector saltos: conversión flotante-entero |
| ADRAY2 | 0005-0006 | 5-6 | Vector saltos: conversión flotante-entero |
| CHARAC | 0007 | 7 | Carácter búsqueda |
| ENDCHR | 0008 | 8 | Flag: rastrea comillas de fin de variable en serie |
| TRMPOS | 0009 | 9 | Col. pantalla desde últ. TAB |
| VERCK | 000A | 10 | Flag: 0=carga, 1=verificar |
| COUNT | 000B | 11 | Puntero buffer entrada / n.° subscritos |
| DIMFLG | 000C | 12 | Flag: DIMensión tabla por defecto |
| VALTYP | 000D | 13 | Tipo datos : $FF=serie,$00=número |
| INTFLG | 000E | 14 | Tipo datos : $80=entero,$00=flotante |
| GARBFL | 000F | 15 | Flag: busca DATA/comillas LIST/recolector de basura |
| SUBFLG | 0010 | 16 | Flag: refer. suscrito/llamada función usuario |
| INPFLG | 0011 | 17 | Flag:$00=INPUT,$40=GET, $98=READ |
| TANSGN | 0012 | 18 | Flag: signo TAN/Resultado comparación |
| | 0013 | 19 | Flag: interrogación de INPUT |
| LINNUM | 0014-0015 | 20-21 | Temp: valor entero |
| TEMPPT | 0016 | 22 | Puntero: pila temporal de serie |



Marcus Wilson-Smith

**Movimiento de apertura**
Tradicionalmente, el juego del *go* se desarrolla sobre un tablero de madera que tiene grabada una cuadrícula de 19 por 19 líneas que se intersectan. Para nuestra versión informatizada hemos reducido el tamaño del tablero a una cuadrícula de 15 por 15, para permitir que se pueda visualizar cómodamente en una pantalla

# Jugadas preliminares

## Cuando se escribe un programa para un juego como el «go», es mejor empezar por las rutinas de E/S

El programa de *go* se presentará en cuatro versiones separadas: para los micros BBC Modelo B, Commodore 64, Sinclair Spectrum y Amstrad CPC 464/664. Siempre que ha sido posible, los números de línea se han mantenido constantes para todas las versiones, pero por diversas razones ha sido necesario incluir líneas extras para implementar algunas rutinas en cada una de las máquinas. Esto es especialmente evidente en el caso de la recursión empleada en el BASIC BBC (no está implementada la recursión en ninguna de las otras tres máquinas). Por consiguiente, es necesario implementar una pila para el usuario con el fin de conseguir que los listados sean lo más parecidos posible. Cuando el texto hace mención a rutinas y variables del listado, éstas corresponden a la versión para el BBC Micro, si bien las correspondencias entre los cuatro listados son bastante directas.

La siguiente sección del programa se ocupa de la inicialización de variables, la creación de la visualización del tablero y las rutinas de entrada. Las líneas de la 10 a la 140 constituyen el cuerpo principal del programa. Antes de entrar el bucle del juego propiamente dicho (de la línea 60 a la 90), se llamará a las rutinas PROCinicializar y PROCintroduccion.

PROCinicializar sólo se utiliza en la primera ejecución del programa, para DIMensionar el tablero (tablero%), inicializar el cursor, etc. El tablero se DIMensiona para estar compuesto por una serie de 255 bytes, creando una superficie de juego de 15 por 15 (en lugar del tablero normal del *go,* de 19 por 19). Esto obedece a varias razones, de las cuales la más importante es la velocidad. No se puede esperar que un programa de juego en BASIC opere con especial rapidez, pero reduciendo las dimensiones del tablero el tiempo de ejecución se acortará en aproximadamente un tercio, sin incidir en la forma de jugar al juego. Por supuesto, esto también significa que el tablero cabrá limpiamente en una página (256 bytes, un cuarto de byte) de memoria, permitiendo un límite compuesto por cuadrados individuales. Asegurando que todas las referencias al tablero estén comprendidas en la escala entre 0 y 255, este reborde circundará el tablero.

Las variables inicializadas en las líneas 190 y 200 son constantes, pero el uso del nombre de la variable en lugar de números simplifica mucho las modificaciones. Por ejemplo, si usted quisiera un juego para dos personas, o bien que el ordenador jugara consigo mismo, sólo necesitaría cambiar los valores de blancas% y negras% para poder utilizar normalmente todas las rutinas. Sobre la base de estos valores, cada byte del tablero se empleará de la forma que se indica en el diagrama.

PROCintroduccion se utiliza al inicio de cada partida, para preparar el contador de movimientos (movimientos%), la condición de final (fin%), etc. Hace uso de las rutinas PROCinic_juego y PROCpantalla_titulos. La pantalla de títulos es bastante elemental debido a las restricciones de espacio para los listados, pero usted está en libertad de modificarla en la medida que así lo desee. Por ejemplo, podría añadir el símbolo japonés del *go* o, quizá, algunos acordes de música oriental. Si desea añadir trozos de código, puede emplear para ello todos los números de línea a partir del 5000.

PROCimprimir_tablero, como es natural, imprime el tablero. FNint_to_char se utiliza desde esta rutina para convertir una coordenada de tablero en enteros en las coordenadas de caracteres que emplea la visualización en pantalla; se DEFine en la línea 2260.

FNinput y PROCmensaje son dos rutinas de propósito general para la entrada del usuario y la impresión de mensajes, basadas en la matriz mens$ inicializada en PROCleer_mensajes. Esta matriz retiene 10 avisos o mensajes que el ordenador necesita visualizar durante el juego. El mensaje apropiado se

selecciona pasando a estas dos rutinas el número del elemento de la matriz correspondiente, M%. La incorporación de estas rutinas, en lugar de limitarse a solicitar y recibir una entrada y una salida cuando es necesario, aumenta la flexibilidad del programa.

Al digitar la versión para el Spectrum, puede encontrarse con un problema entre las líneas 1480 y 1505. Los números de estas líneas subrayados hacen referencia a los gráficos de símbolos del Spectrum. Usted los obtendrá pulsando primero Caps Shift y 9 juntas, entrando por tanto en modalidad de gráficos (el cursor G) y después digitando los números indicados. Pulse nuevamente Caps Shift y 9 para salir de la modalidad de gráficos al final de la sentencia de impresión. Si el número va precedido por sh, entonces digite el número mientras pulsa la tecla Shift al mismo tiempo.

Si ejecuta el código tal como está, obtendrá una pantalla de títulos pidiéndole una cantidad de fichas de handicap, seguida por la pantalla principal del juego. Luego el programa hará indefinidamente un bucle. Por el momento el número de handicap se ignora, pero en el próximo capítulo lo añadiremos, así como algunas otras rutinas generales.

## Variables del juego del «go»

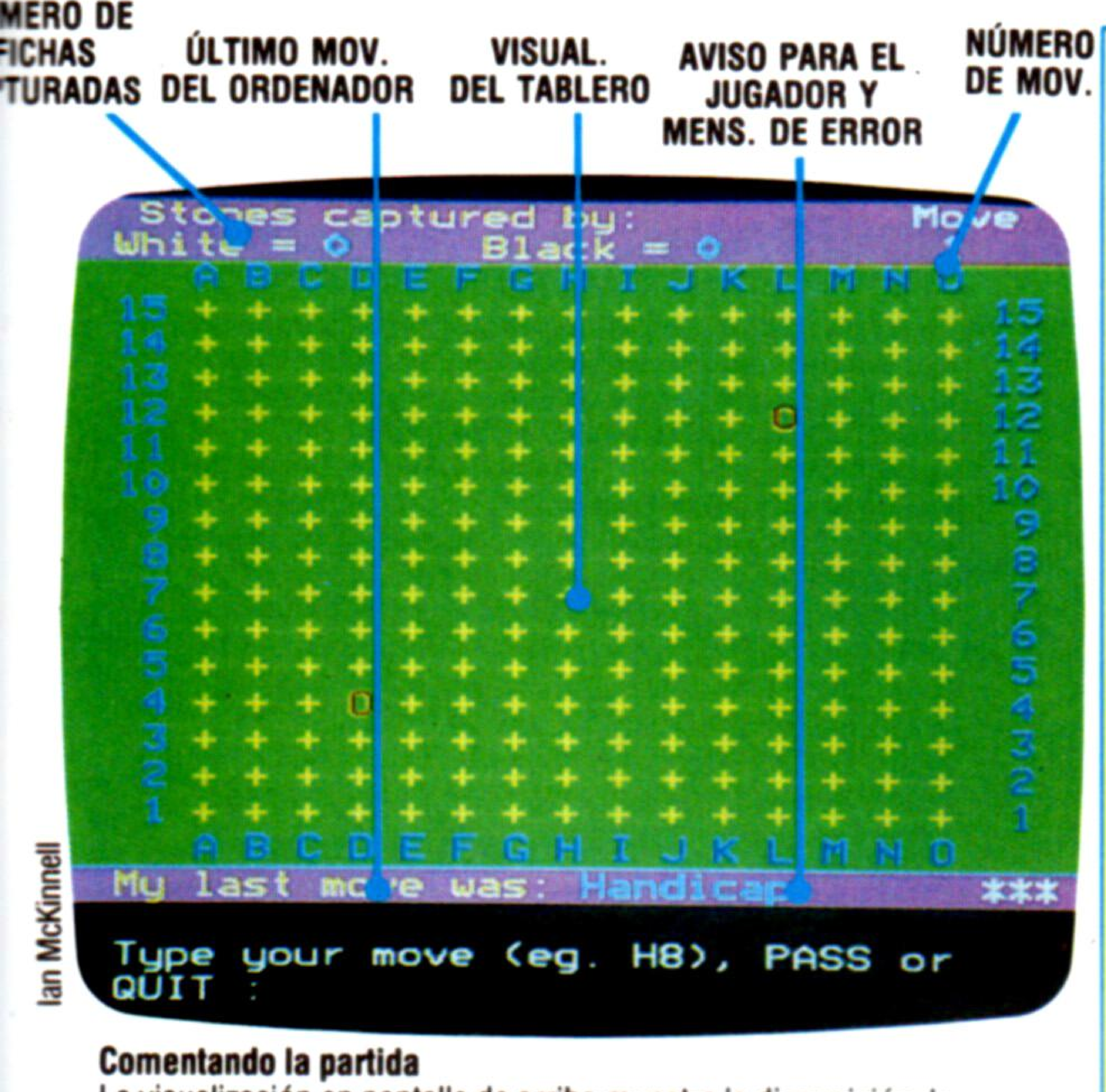
| Variable | Finalidad |
|---|---|
| **negras%** | Valor 1: ficha negra en byte del tablero. |
| **tablero%** | El valor de comienzo para el tablero de 256 bytes en la memoria. |
| **captura%(2)** | Retiene la cantidad de fichas capturadas por las negras y las blancas. |
| **color%** | Valor 3. Se utiliza para enmascarar los bits de color en un byte del tablero. |
| **dir%(4)** | Retiene los desplazamientos necesarios para avanzar un cuadrado hacia el N, E, S y O. |
| **licencia%** | Valor 8. Se usa en la rutina de búsqueda de grupos para marcar licencias ya contadas. |
| **marcador%** | Valor 4. Se usa en las rutinas de búsqueda de grupos para marcar fichas ya contadas. |
| **movimiento%** | Retiene la cantidad de movimientos efectuados en cada partida para la visualización. |
| **blancas%** | Valor 2: ficha blanca en byte del tablero. |
| **atari1$** | Retiene 5 espacios cualesquiera o «Atari» si el último mov. del ord. produjo esta sit. |
| **atari2$** | Retiene 5 espacios cualesquiera o «Atari» si el último mov. del jugador produjo esta situación. |
| | Nota: Atari es la acción de colocar una ficha de modo tal que deje a uno o más grupos del oponente con una sola licencia. |
| **eje$** | Parte de la visualización del tablero. |
| **mens$** | Retiene todos los mensajes generales de E/S, utilizados por PROCinput y PROCmensaje. |

## Módulo Uno

**BBC Micro:**

```
10 MODE 7
20 *FX14,6
30 PROCinicializar
40 PROCintroduccion
50 CLS: PROCimprimir_tablero
60 movimiento%=movimiento%+1: REM aqui movimientos de las
   blancas
70 IF fin% GOTO 100
80 movimiento%=movimiento%+:REM aqui movimientos de las
   negras
90 IF NOT fin% GOTO 60
100 resp$=FNinput (21,9,1)
110 IF resp$=«S» GOTO 40 ELSE IF resp$<>«N» GOTO 100
120 PROCmensaje (22,5,"")
130 PRINT: END
140 :
150 REM *********************************************
160 :
170 DEF PROCinicializar
180 LOCAL L%
190 negras%=1 : blancas%=2 : color%=3
200 marcador%=4 : licencia%=8
210 @%=2
220 DIM tablero% 255
230 VDU 23;8202;0;0;0;
240 DIM captura%(2)
250 eje$=CHR$130+CHR$157+CHR$132+"A B C D E F G H I J K
   L M N O"
260 PROCleer_mensajes
290 DIM dir%(4)
300 RESTORE 340
310 FOR L%=1 TO 4
320    READ dir% (L%)
330    NEXT
340 DATA 16,1,-16,-1
350 ENDPROC
360 :
370 REM *********************************************
380 :
390 DEF PROCleer_mensajes
400 LOCAL M%
410 RESTORE 460
420 DIM mens$(9)
430 FOR M%=0 TO 9
440    READ mens$(M%)
450    NEXT
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "Entrada ilegal: "
480 DATA "Ficha ya en ese lugar: "
490 DATA "Ilegal. Ko en ese lugar: "
500 DATA "Ilegal. Suicidio en ese lugar: "
510 DATA "    O.K. JUEGO TERMINADO."
520 DATA ""
530 DATA "    Cuantas fichas puedo tener de handicap
   (2-9) ?"
540 DATA "Digite su movimiento (por ej. H8), PASO o
   ABANDONO :"
550 DATA "Quieres jugar otra partida (S/N)?"
560 ENDPROC
570 :
580 REM *********************************************
1260 :
1270 DEF PROCintroduccion
1280 PROCinic_juego
1290 PROCpantalla_titulos
1300 ENDPROC
1310 :
1320 REM *********************************************
1330 :
1340 DEF PROCinic_juego
1360 atari1$="     ":atari2$="     "
1370 posicion%=0:movimiento%=1
1380 fin%=FALSE
1390 captura%(1)=0:captura%(2)=0
1410 ENDPROC
1420 :
1430 REM *********************************************
1440 :
1450 DEF PROCpantalla_titulos
1460 CLS
1470 PRINT TAB(12,4);CHR$145;CHR$154;"h7+$
   h7k4"
1480 PRINT TAB(12,5);CHR$145;CHR$154;"j5k5
   j5j5"
1490 PRINT TAB(12,6);CHR$145;CHR$154;"pssp
   pssp"
```






Ian McKinnell

**Comentando la partida**
La visualización en pantalla de arriba muestra la disposición de las diversas características del juego. Además de mostrar el tablero, el programa informa al jugador sobre el número de fichas capturadas tanto por el ordenador como por el jugador, el número de movimientos realizados en la partida, y cuáles fueron los últimos movimientos efectuados por la máquina y por su adversario. La porción inferior de la pantalla está reservada a avisos y mensajes de error dirigidos al jugador. A medida que se desarrolle el juego veremos cómo el programa detecta y desautoriza movimientos ilegales como el «suicidio» o cualquier intento por colocar fichas en posiciones ya ocupadas




**Bits a bordo**
Cada posición del tablero se representa en la memoria mediante un único byte. En el diagrama superior vemos el trazado de los 255 bytes que representan todas las intersecciones en el tablero de 15 por 15. Los bits de cada byte retienen información sobre el estado de la intersección correspondiente del tablero. Los dos bits menos significativos determinan el color de la ficha presente (o indican si la intersección está libre). Los bits 2 y 3 los utilizarán las rutinas de evaluación del «estado del juego», que desarrollaremos en próximos capítulos

```
1500 PRINT TAB(9,10);CHR$134;"por Marcus
     Jeffery"
1510 PRINT TAB(1,13);CHR$133;«Jugaras con
     las";
1520 PRINT CHR$135;"fichas";CHR$133;"blancas,
     y"
1530 PRINT CHR$133;"el ordenador (al ser mas
     débil!)"
1540 PRINT CHR$133;"jugara con las";CHR$129;"fichas";
     CHR$133;"negras con ventaja de"
1550 PRINT TAB(13);CHR$133;"handicap"
1560 hand%=VAL(FNinput (20,7,1))
1570 IF hand%<2 OR hand%>9 THEN 1560
1590 ENDPROC
1600 :
1610 REM **********************************************
1720 :
1730 DEF PROCimprimir_tablero
1740 LOCAL P%, X%, Y%
1750 PRINT TAB(0,0);CHR$133;CHR157;CHR$131;"Fichas
     capturadas por:";
1760 PRINT TAB(32,0);CHR$135;"movimiento"
1770 PRINT TAB(0,1);CHR$133;CHR$157;CHR$131;"Blancas
     =";CHR$134;captura%(2);
1780 PRINT TAB(16,1);CHR$131;"Negras
     =";CHR$134;captura%(1);
1790 PRINT TAB(33,1);CHR$135;movimiento%
1800 PRINT TAB(0,2); eje$
1810 FOR Y%=15 TO 1 STEP -1
1820    PRINT TAB (0,18-Y%);CHR$130;CHR$157;CHR$132;
     RIGHT$("  "+STR$ (Y%),2);
1830    FOR X%=1 TO 15
1840    P%=tablero%?(16*Y%+X%)
1850    IF P%=1 THEN PRINT CHR$129;"O";:GOTO
        1880
1860    IF P%=2 THEN PRINT CHR$135;"O";:GOTO
        1880
1870    PRINT CHR$131;"+";
1880    NEXT
1890    PRINT TAB(35,18-Y%);CHR132,Y%
1900    NEXT
1910 PRINT TAB(0,18);eje$
1920 PRINT TAB(0,19);CHR$133;CHR$157;CHR$131;"Mi ultimo
     movimiento fue:";
1930 PRINT CHR$134;FNint_to_char(posición%);TAB(30)
     atari1$
1940 PRINT TAB(16,22);atari2$;CHR$7
1950 ENDPROC
1960 :
1970 REM **********************************************
1980 :
1990 DEF FNinput(P%,M%,W%)
2000 LOCAL S%,A$,I$
2010 PRINT TAB(39,P%+1);STRING$(119,CHR$127);
2020 A$=""
2030 PRINT TAB(3,P%);mens$(M%);CHR$134;
2040 PRINT STRING$(W%,CHR$32);STRING$(W%,
     CHR$127);
2050 *FX21,0
2060 I$=GET$ : IF ASC(I$)=13 THEN GOTO 2120
2070 IF ASC(I$)<>127 GOTO 2090
2080 IF S%>0 THEN S%=S%-1:A$=LEFT$(A$,S%):GOTO 2110
     ELSE GOTO 2060
2090  IF I$>="a" AND I$<="z" THEN I$=CHR$(ASC(I$)
     -32)
2100  IF S%<W% THEN S%=S%+1:A$=A$+I$ ELSE GOTO
     2060
2110 PRINT I$; : GOTO 2060
2120 =A$
2130 :
2140 REM **********************************************
2150 :
2160 DEF PROCmensaje(P%,M%,A$)
2170 LOCAL L%
2180 FOR L%=P% TO P%+1
2190    PRINT TAB(39,L%);STRING$(39,CHR$127);
2200    PRINT CHR$141;CHR$136;CHR$129;mens$(M%);
        A$;
2210    NEXT
2220 ENDPROC
2230 :
2240 REM **********************************************
2250 :
2260 DEF FNint_to_char (P%)
2270 IF P%=0 THEN ="Handicap"
2280 =CHR$(P% MOD 16 + 64)+STR$(P% DIV 16)+" "
2290 :
2300 REM **********************************************
```

# Plan de acción

## Finalmente veremos cómo actúan dos importantes paquetes: «Sycero» y «The Last One»

Todo el software destinado a posibilitar que el principiante absoluto genere aplicaciones para ordenador debe satisfacer algunos puntos rigurosos:

- Debe favorecer un enfoque *de arriba abajo,* en el que el usuario pueda definir el proyecto a realizar y especificar con claridad cada punto a medida que vaya progresando el trabajo.
- Debe ser activado por menú, con solicitudes de opciones múltiples que vayan guiando al usuario a través de las opciones disponibles.
- Debe detectar los errores apenas se produzcan y permitir cambios o correcciones simples.
- El código resultante debe ser «transportable», es decir, debe poder ser ejecutado en otro ordenador, con independencia del software del sistema que lo generó.
- Debe producir amplia documentación del proceso de generación del programa, de modo que el usuario que no esté familiarizado con la programación precedente pueda introducir correcciones o mejoras.

Sin embargo, puesto que el generador de programas es una herramienta cuyo uso no está limitado a los no iniciados, también ha de resultar aceptable al usuario más experimentado. Es probable que un programa dirigido a un nivel muy bajo moleste e incluso confunda al usuario más avanzado, además de demorar las cosas con medidas de protección y seguridad innecesarias. También es probable que el usuario más experimentado tenga algunas rutinas favoritas (formas de usar el BASIC, p. ej.) y el generador habrá de estar preparado para aceptarlas. Teniendo en cuenta todas estas consideraciones, los dos generadores de programas de mayor aceptación, *The Last One (TLO)* y el más reciente *Sycero,* obtienen altas calificaciones.

Ambos utilizan ampliamente menús informativos. *Sycero* posee pantallas de ayuda para los controles del cursor y gráficos; las instrucciones para edición de textos están disponibles en todo momento pulsando las teclas Control y H. De los dos sistemas, el *TLO* es el que sigue con más fidelidad el enfoque de arriba abajo. Comienza por elaborar un diagrama de flujo, que en realidad es una serie de comentarios (REM) acerca de lo que se pretende que haga cada sección del programa (Branch on 4-option menu, p. ej.) y, si así se desea, se lo puede incorporar al final del programa en BASIC.

Las pantallas de entrada y de visualización se definen durante el proceso de codificación del programa y el código resultante estará en formato ASCII. El usuario entonces ha de salir del generador, cargar (LOAD) el programa generado y volver a guardarlo (SAVE) en código binario. En el *Sycero,* el usuario se encarga en primer lugar de la definición de archivos y el diseño de la pantalla. Tras ello vienen el «proceso de variables de entrada» (asociando avisos de ayuda y mensajes de error a la entrada del usuario en cada caso), las definiciones de listados y otras varias opciones. Los diversos módulos se unirán entre sí justo antes de la codificación.

Aunque este enfoque es menos amable para el usuario inexperto, ofrece la ventaja de que exige definir cuidadosamente el proyecto antes de codificarlo, ya que de lo contrario no se ejecutará en absoluto. Además, si durante la codificación el generador detecta algún error, detiene el proceso y le informa al usuario acerca del error, de modo que pueda enmendarlo. El proceso de codificación concluye guardando el programa, en archivo binario, que se puede ejecutar desde dentro del entorno del *Sycero.*

Los dos generadores producen un código completamente transportable. De hecho, si el programa *TLO* es suficientemente corto, es posible ejecutar la versión ASCII en una máquina de estándar MSX (en la medida en que se ajuste a las limitaciones de memoria del micro). No obstante, ambos programas incluyen exhaustivas rutinas de detección de errores en los programas que generan, con lo que resultan bastante voluminosos.

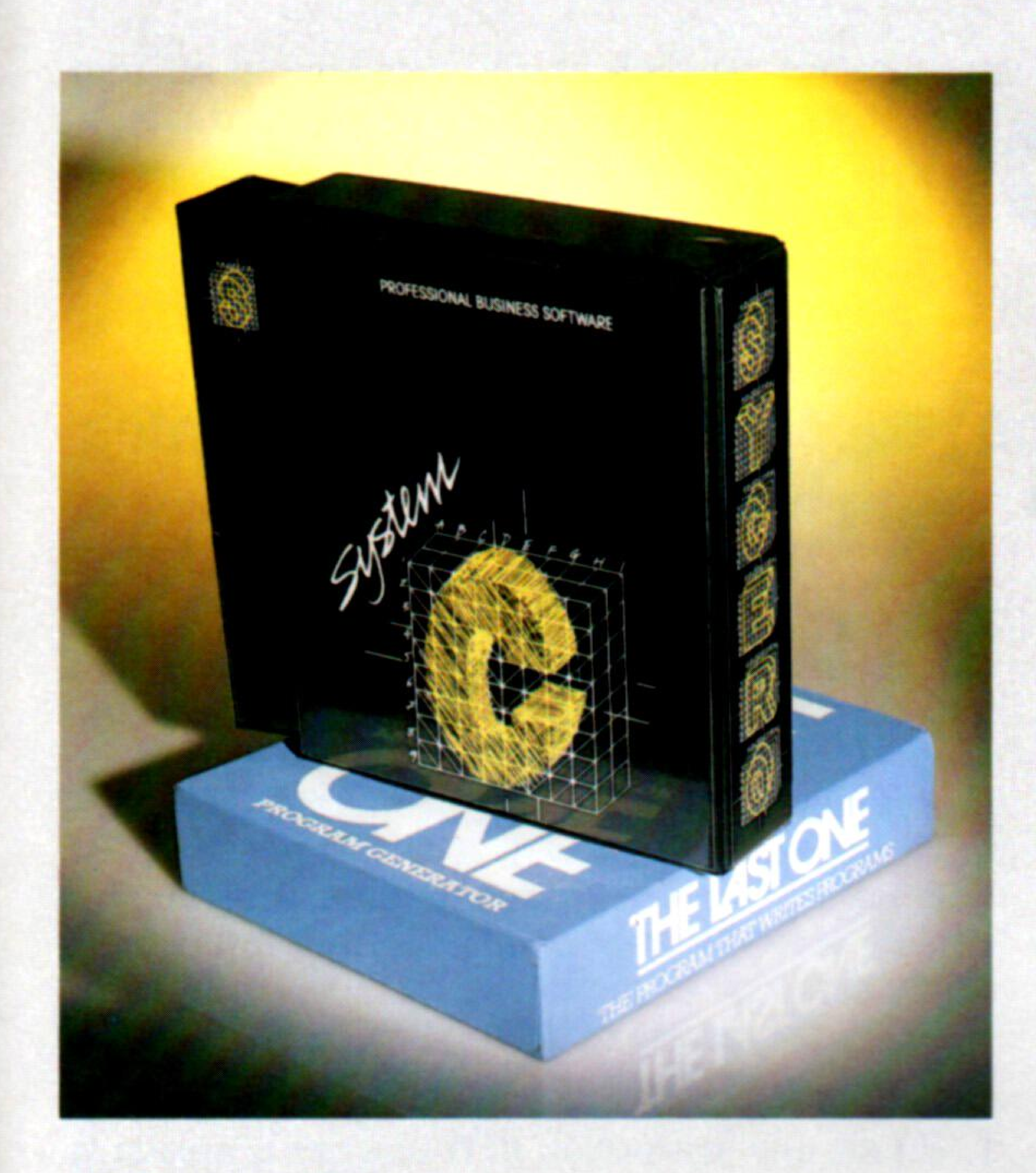
Como es común en el software MS-DOS, tanto el *TLO* como el *Sycero* han de estar configurados con el hardware en uso con los programas Install, que se pueden utilizar con máquinas de disco rígido o disco flexible de unidad doble. El *TLO* también debe «certificar» un disco de trabajo, un proceso en cierto modo similar al formateo. A este fin, un disco de trabajo puede ser un disco flexible entero o bien un subdirectorio del disco rígido (este último se certifica como parte del proceso de instalación). La certificación es esencial, porque si durante la corrección del diagrama de flujo el programa no puede detectar una zona de disco de trabajo, será incapaz de hallar el diagrama de flujo en cuestión. Esto significa que el usuario tendrá que volver al principio o bien intentar manipular el código en BASIC. Como siempre, puede protegerse contra esta clase de desastres haciendo frecuentes copias de seguridad del contenido del disco que esté utilizando.

Tanto el *TLO* como el *Sycero* generan una profusa documentación, listando cuestiones tales como las características del diseño de pantalla, los nombres de las variables utilizadas, etc. *Sycero* añade la fecha y la hora en la pantalla, así como en las copias impresas, para que se puedan distinguir, al comprobarlas, las primeras versiones de las posteriores mejoradas.

Tras la instalación, al usuario del *TLO* se le ofrece un Main Dispersal Menu (menú de dispersión principal) de ocho opciones:

| | |
|---|---|
| Crear un programa | Interrogación |
| Modificar un programa | Certificar disco nuevo |
| Modificar un archivo | Resumir codificación |
| Definir un archivo | Retornar al BASIC |

Tras elegir la primera opción y responder en sentido afirmativo a la pregunta: ¿Requerirá archivos su programa?, el usuario ha de definir los archivos ne-




**Dos "escritores" de programas**
Tanto *The Last One (TLO)* como *Sycero* permiten que el usuario genere programas en BASIC, que se ejecutan con independencia del ordenador anfitrión. Sin embargo, la definición precisa de las necesidades del usuario implica una concienzuda preplanificación antes de que ambos paquetes se puedan utilizar de forma rentable

Ian McKinnell

cesarios con sus campos y tipos de campos (alfabéticos, numéricos o datos). Después de esto, la pantalla visualizará el Flowchart Creation Menu (menú de creación de un diagrama de flujo). En la versión más reciente, hay 20 opciones disponibles, numeradas del 1 al 12 y del 14 al 21 (evidentemente, el autor del *TLO* es supersticioso):

| | |
|---|---|
| Listar diagrama de flujo | Borrar |
| Modificar diagrama flujo | Establecer punteros de arch. |
| Codificar programa | Leer un archivo |
| Mezclar diagramas flujo | Escribir un archivo |
| Abortar | Buscar o clasif. un arch. |
| Entrada por teclado | Mezclar |
| Visualizar datos | Comprobación de reg. |
| Bifurcaciones | Borrar archivo |
| Cálculos | Funciones de base datos |
| Funciones especiales | Multifunciones |

Utilizando estas opciones se puede desarrollar un diagrama de flujo, que podría parecerse a lo siguiente, para un archivo de nombres y direcciones:

1. . Bifurcación a un menú de 3 opciones
2. . Establecer puntero en el final del arch. Direcciones
3. . Entrada por teclado para archivo Direcciones
4. . Escribir datos en archivo Direcciones
5. . Preguntar <¿Has terminado?>. Bifurcar si «No»
6. . Dirigir bifurcación incondicional
7. . Bifurcación a un menú de 3 opciones
8. . Establecer puntero al comienzo del arch. Dirs.
9. . Búsqueda por teclado de archivo Direcciones
10. . Visualizar datos de archivo Direcciones
11. . Preguntar <¿Otra búsqueda?>. Bifurc. si «Sí»
12. . Dirigir bifurcación incondicional
13. . Clasificar archivo Direcciones
14. . Establecer puntero al comienzo del arch. Dirs.
15. . Leer datos de archivo Direcciones
16. . Visualizar datos de archivo Direcciones
17. . Dirigir bifurcación incondicional
18. . Terminar

A B C D E F G H I J K L

Entonces puede empezar la codificación. Hasta este punto, el usuario no ha podido salir del programa sin perder todo el trabajo realizado hasta ese momento. Sin embargo, cuando comienza la codificación se podrá salir y usar la opción Resumir Codificación para volver a comenzar. Durante la codificación, se completan los destinos de las bifurcaciones; el primer menú, por ejemplo, se bifurcará a 2 (escribir datos), 7 (leer datos) o 18 (terminar), y se diseñan las pantallas. Se pueden guardar (SAVE) pantallas, lo que es aconsejable, puesto que se las puede modificar para utilizarlas ulteriormente en cualquier lugar del programa o incluso en otros programas. Sin embargo, ni en el software ni en el manual se le da a esta facilidad la importancia que requiere. En el *Sycero,* las pantallas se guardan (SAVE) automáticamente.

## Menú de apertura

El menú de apertura del *Sycero* ofrece 13 opciones que se utilizan más o menos por el orden en que aparecen:

Configuración del sistema
Inicialización
Definición de archivo-campos del sistema
Definición de la pantalla
Proceso de la pantalla
Definición de listado
Proceso de listado
Definición del programa
Generar un programa
Crear un archivo de datos «vivos»
Ejecutar un programa generado
Utilidades
Finalizar la sesión

Cuando se llegue a la opción de generación del programa ya se habrá hecho gran parte del trabajo, y el proceso relativamente simple que sigue exigirá poca intervención, a menos que el software detecte un error obvio. Por supuesto, el sistema no puede deducir sus intenciones, de modo que es imperativo que lo instruya de una manera *precisa.*

La principal diferencia entre los dos generadores es de enfoque. Como se afirma en el propio manual del *Sycero,* «la mejor forma de desarrollar un sistema es utilizar el enfoque de arriba abajo. Comience con un boceto lo más general posible y luego, de forma gradual, vaya rellenando con los detalles desde arriba hacia abajo. Lamentablemente, la mejor forma de entrar los detalles en un generador es la contraria, de abajo arriba».

Al comenzar con un generador de diagrama de flujo, el *TLO* se aproxima más al enfoque clásico de abajo arriba, aunque paradójicamente puede ser que, de este modo, estimule al usuario a comenzar el trabajo con una preplanificación inadecuada. Por el contrario, es virtualmente imposible trabajar con *Sycero* a menos que se haga una preplanificación, si bien hace que resulte más fácil ir revisando las cosas a medida que uno va trabajando.

Tanto *Sycero* como *TLO* ofrecen al usuario una forma conveniente de generar programas en BASIC que se ejecuten independientemente de la máquina anfitriona. A pesar de sus limitaciones, pueden resultar útiles para la construcción de utilidades de bases de datos y programas sencillos de cálculo con preguntas y respuestas.

# Tener gancho

## La Interface 1 para el Spectrum nos proporciona unas interesantes rutinas que vamos a analizar aquí

La Interface 1 tiene ocho Kbytes de memoria ROM, y ocupa la primera sección de la memoria. Proporciona las rutinas necesarias para gestionar los dispositivos adicionales (como los microdrives); asimismo amplía el intérprete del BASIC para que acepte instrucciones como CAT y FORMAT. Esta ROM suele denominarse ROM Sombra (Shadow ROM).

Hay numerosas versiones diferentes de esta ROM, y existen diferencias de importancia entre la implementación inicial (versión 1) y la actualmente presentada (versión 2). La versión 1 se mostró muy poco eficaz en su empleo de microdrives, y diversas operaciones de las que ofrecía no las podía realizar.

Las versiones siguientes de la ROM son más eficaces en el manejo de los microdrives, además de permitirnos emplear más espacio en un cartucho de microdrive. Asimismo fueron depurados los errores y se incorporaron nuevas facilidades relacionadas con las impresoras en serie.

Estas diferencias de ROMs suelen ser «invisibles» para el usuario, mientras sólo se acceda a la Interface mediante BASIC o con las técnicas consagradas de programación en código máquina. Sin embargo, si usted intenta llamar a la ROM Sombra directamente, con toda probabilidad encontrará dificultades, ya que las rutinas están colocadas en direcciones que difieren según la versión de ROM. Las direcciones específicas de ROM Sombra que damos se refieren a la ROM versión 1.

Pero ¿cómo puede la ROM Sombra ocupar los ocho primeros Kbytes de la memoria si la ROM del BASIC se encuentra justamente en esta zona? La técnica usada es similar al sistema paginado de ROM que emplea el BBC Micro. Siempre que el Z80 vaya a leer una instrucción de las direcciones &08 o bien &1708 con la Interface 1 conectada, la ROM del BASIC es «paginada» y relevada por la ROM Sombra en las operaciones siguientes. Hay una rutina en la ROM Sombra que vuelve a restaurar la ROM del BASIC en su momento.

Antes de que esto suceda, sin embargo, debe inicializarse la Interface 1. Esto se logra después de:

1. Emitir una instrucción NEW una vez acoplada la Interface.
2. Paginar por primera vez la ROM Sombra.

El resultado práctico de este proceso de inicialización es que se ha establecido un nuevo conjunto de variables de sistema. Éstas ocupan la RAM a partir del final de las antiguas variables de sistema y hasta el área de los canales: más de 50 bytes en total. Lo anterior, junto con el establecimiento del mapa del microdrive y demás información necesaria para los distintos dispositivos administrados por la Interface 1, hace que, conectada ésta, el inicio del texto de programas en BASIC se desplace hacia arriba en la memoria. Por esto es muy poco recomendable almacenar el código máquina en las sentencias REM de la línea 1 después de haber conectado la interface (ya que la línea 1 del programa no tiene ahora una posición constante en la memoria).

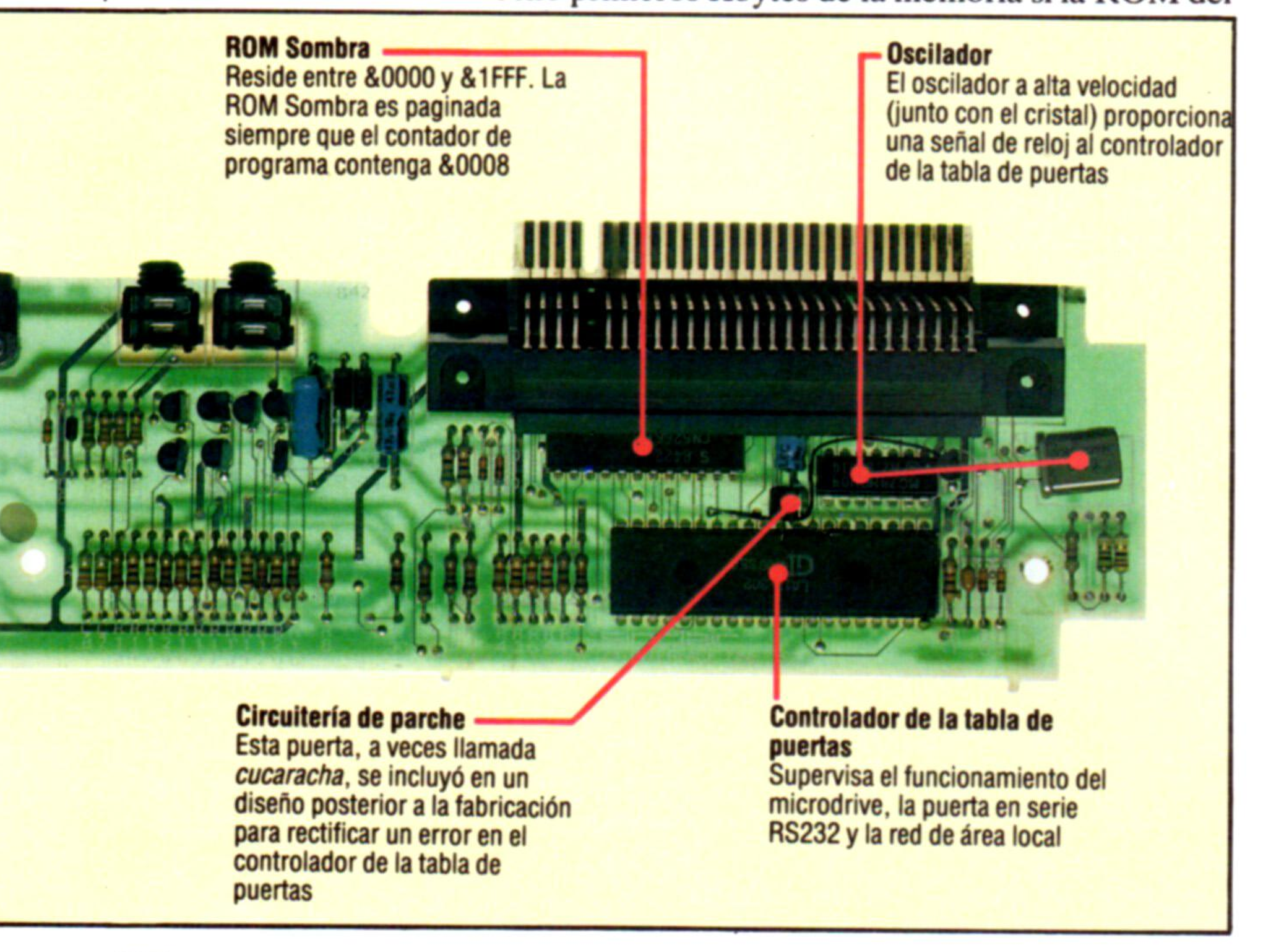
Como ya hemos indicado, la Interface 1 ofrece también instrucciones en BASIC adicionales. Incluso podemos añadir algunas de elaboración propia, como veremos en otro capítulo. Nos limitaremos aquí a examinar someramente la implementación de instrucciones como CAT y FORMAT. En un Spectrum no ampliado, eludirán toda comprobación llevada a cabo por el intérprete del BASIC, por lo que generará normalmente un error. Esto se hace mediante la instrucción RST &08, que accede a la dirección &08. Cuando la Interface 1 es conectada, esta instrucción paginará la ROM Sombra, que proporcionará al enojado fragmento del BASIC unas «segundas gafas» para que pueda ver si se trata de una instrucción que pueda interpretar correctamente. Y si puede, lo hará. De esta manera más o menos funciona una instrucción como la siguiente, que guarda el programa de nombre *Juan* en el microdrive:

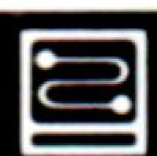
```
SAVE *"m";1;"Juan"
```

La parte * de la instrucción genera una condición de error, obligando al control a pasar a la dirección &08, y por tanto a la ROM Sombra, donde tiene lugar la correcta interpretación de la sentencia.

Veamos ahora las rutinas adicionales en lenguaje máquina que incorpora la Interface 1. Examinare-

**La procesión por dentro**
El dibujo muestra los principales componentes de la placa de circuito impreso correspondiente a la Interface 1. No sólo proporciona facilidades para microdrive, interface serial y LAN, sino que permite la creación de nuevas instrucciones en BASIC definidas por el usuario. En otro capítulo examinaremos estas instrucciones extras definidas por el usuario



mos las que pueden dar en llamarse rutinas para «tareas universales», que son las rutinas no relacionadas con el control del microdrive, la interface serial o las facilidades LAN.

# Códigos de enganche

El primer problema es llamar a estas rutinas. La ROM Sombra sólo se activa después de una llamada a alguna de las direcciones anteriormente mencionadas. Esto se resuelve llamando a la dirección &08 mediante las siguientes instrucciones:

```
RST     &08
DEFB    nn
```

donde nn es un valor entre &1B y &32 (los valores fuera de este intervalo generan error). A estos valores nn se les denomina *códigos de enganche (hook codes)*, y cada una de estas llamadas pertenece a una rutina diferente de la ROM Sombra. Los efectos de los códigos de enganche no han cambiado en las distintas versiones de la ROM Sombra. Antes de examinar algunas de estas llamadas, advertiremos varias cosas.

1. Las operaciones del código de enganche suelen afectar a todos los registros, por lo que es necesario proteger los registros que puedan necesitarse después. Además, siempre es bueno guardar el par de registros HL, pues es esencial a la hora de volver al BASIC.
2. Algunas rutinas de los códigos de enganche desactivan las interrupciones. Usted debe reactivarlas si no está seguro.
3. Cuando se usa un código de enganche, ponga el valor &5C3A en el par de registros IY.
4. Antes de emplear un código de enganche, se recomienda asegurarse de que las variables de sistema de la Interface 1 han sido establecidas. Existe un código de enganche que hará esto y vamos a examinarlo antes de nada.

• *Código de enganche 49 (&31):* se encarga de crear, o insertar, las variables de sistema de la Interface 1. Si usted no está seguro de que éstas hayan sido creadas, deberá hacer la siguiente llamada antes de emplear cualquier otro código de enganche:

```
RST     8
DEFB    49
```

Una vez que la ROM Sombra ha ejecutado una rutina de código de enganche, el control pasa a la ROM principal. La ROM Sombra también nos proporciona algunas rutinas que están presentes en la ROM principal del BASIC de una manera menos adecuada. Éstas tratan de entradas/salidas a pantalla desde el teclado.

• *Código de enganche 27 (&1B):* provoca una espera hasta que se digita en el teclado un carácter, y pone en el registro A el código del carácter correspondiente a la tecla pulsada (obsérvese que esto no aparece en pantalla).

• *Código de enganche 32 (&20):* inspecciona el teclado en el momento de ser llamado e indica (por medio del estado del flag C) si se ha pulsado una tecla o no. Esta rutina no espera hasta que se pulse una tecla. Si ha sido pulsada, el flag C se pone a 1, y en caso contrario a cero. Es fundamental que se activen las interrupciones antes de llamar a los códigos de enganche 27 y 32, dado que la inspección del teclado en el Spectrum se gestiona mediante interrupciones.

• *Código de enganche 28 (&1C):* se encarga de la impresión de un carácter (que conoce gracias al código ASCII previamente contenido en el registro A) en la corriente 2, que suele ser la parte superior de la pantalla de televisión. El empleo de esta rutina, junto con la del código 27, se ilustra en el siguiente fragmento en lenguaje máquina. Si usted utiliza este programa, observe que es un bucle indefinido, y que tendrá que desconectar el ordenador para salir de él.

```
            ;llama rutina ROM Sombra - ponga atención!
210000      ld      hl,ADDRESS          ;inserta direccion de rutina
22ED5C      ld      (23789),hl          ;pone direccion en variable de sis-
CF          rst     #08                 tema
32          defb    #32                 ;opera
```

• *Código de enganche 31 (&1F):* es semejante al código 28, sólo que escribe el carácter (cuyo código ASCII está en el registro A) en la corriente 3, que suele ser la impresora ZX.

• *Código de enganche 50 (&32):* se trata del código de enganche final para tareas universales, y lo vamos a examinar con más detención. En las especificaciones del fabricante se le rotula, no sin un cierto misterio, Sinclair Research Use Only (sólo para uso de Sinclair Research). Gracias a él podemos llamar a una rutina de la ROM Sombra a una dirección determinada desde la ROM principal, sin tener que usar códigos de enganche. A causa de las diferencias entre las distintas versiones de la ROM Sombra todo programa que emplee esta llamada para acceder a las rutinas de aquélla se vendrá abajo si se usa en una versión diferente. Los detalles que damos aquí son, pues, a efectos de completitud.

La dirección de la rutina de la ROM Sombra a la que deseamos acceder se encuentra en dos bytes dentro de las variables de sistema mostradas por la Interface 1. La dirección de la variable en cuestión está en las posiciones 23789 y 23790. Seguidamente se realiza el código de enganche, lo que se conseguirá con el siguiente fragmento en código máquina:

```
        ;espera el carácter, imprime en corriente 2
        ;cod. de eng. &1B y &1C
CF    start:  rst   #08        ;establece variables sistema...
31            defb  #31        ;... Interface 1
FB    loop:   ei               ;ei, por si acaso
CF            rst   #08        ;espera un caracter...
1B            defb  #1B        ;... del teclado
CF            rst   #08        ;ahora el caracter esta en A...
1C            defb  #1C        ;... y lo imprime
18F9          jr    loop       ;vuelta hasta el infinito
```

Desde luego que no es una llamada extremadamente útil a menos que se conozcan las funciones de las distintas rutinas de la ROM Sombra y con qué versión de dicha ROM se está trabajando. No obstante, deberemos examinar esta llamada con mayor detención cuando tratemos cómo se emplea la Interface 1 para añadir nuevas instrucciones al BASIC. En el próximo capítulo, de momento, examinaremos aquellos códigos de enganche que son específicos de las operaciones del microdrive.

# Interacción interestelar




**Aventura clásica**
*The hitchhiker's guide to the galaxy* es la última de una larga serie de aventuras de Infocom, de muchísimo éxito, que se inició con la *Trilogía de Zork*, tres juegos llenos de tesoros, magia y misterio que se desarrollaban en el «imperio Zork»

**Una fértil imaginación**
Douglas Adams, autor de *The hitchhiker's guide to the galaxy* (Guía de la galaxia para el autostopista). Escrita originalmente para la radio, desde entonces la serie ha sido publicado por Pan Books, registrando en el mundo unas ventas extraordinarias

Jill Furmanovsky

## ¿Qué diferencias esenciales existen entre un juego basado en disco y otro basado en cassette? Veámosla a través del análisis de un juego de aventuras recién aparecido

**The hitchhiker's guide to the galaxy:** Para la gama de ordenadores Apple, Apricot, IBM y Atari. Pronto habrá una versión para el Commodore 64
**Distribuido por:** Softsel, Softsel House, Sion Gate Way, Great West Road, Brentford, Middlesex, TW18 9DD, Gran Bretaña
**Palanca de mando:** No se necesita
**Formato:** Disco

Los juegos de aventuras requieren enormes cantidades de almacenamiento de datos y son ideales para los sistemas basados en disco, en los cuales las descripciones de nuevos escenarios, mensajes y rutinas de instrucciones se pueden cargar desde disco cuando es necesario. Esta situación siempre ha representado un problema para los programadores de aventuras europeos, quienes han debido escribir juegos para un mercado que tradicionalmente ha rechazado el almacenamiento en disco en favor del realizado en cassette, de menor capacidad. Por este motivo, casi todo el software de aventuras europeo está basado en RAM y, por consiguiente, bastante limitado, aunque los programadores se han vuelto expertos en explotar el limitado espacio disponible. En particular, se ha frenado el desarrollo de «personajes interactivos» (que requieren grandes cantidades de datos), así como la introducción de juegos con abundante vocabulario.

En Estados Unidos, sin embargo, la situación es muy diferente. Todos los micros populares personales tienen fácil acceso a sistemas de disco, y la disponibilidad de ingresos, que registra niveles por lo general más elevados que los de los usuarios europeos, ha significado que tales sistemas por lo general los adquieran los usuarios noveles. El mercado, por lo tanto, ha sido ideal para el desarrollo de complejo software de aventuras o, como algunos prefieren llamarlo, de «ficción interactiva». La casa de software norteamericana Infocom ha sido el líder en juegos para este mercado.

El último producto de Infocom es representativo de los elevados estándares que actualmente se esperan de la compañía. Se trata de *The hitchhiker's guide to the galaxy* (Guía de la galaxia para el autostopista), escrito por Douglas Adams en colaboración con los programadores de Infocom para el Apple IIe y la gama de ordenadores Atari (aparecerá también una versión para el Commodore 64). Exige una única unidad de disco y muchísima paciencia.

El juego se basa, con bastante fidelidad, en la serie escrita originalmente para la radio por Douglas Adams, e incorpora personajes tales como Ford Prefect, Zaphod Beeblebrox, numerosos alienígenas y, por supuesto, Arthur Dent, el antihéroe de la historia, quien un buen día se encuentra arrancado de su existencia suburbana y llevado secretamente a la inmensidad del espacio en la cabina de carga de un Vogon Starcruiser.

El primer punto fuerte del juego, y el más obvio, es el analizador gramatical, la parte del programa que acepta e interpreta la entrada del usuario. Los analizadores gramaticales de Infocom pueden distinguir adjetivos, adverbios y preposiciones, además de los más tradicionales verbos y sustantivos a los que se limitan la mayoría de los juegos de aventuras europeos. Además, el vocabulario es sumamente amplio (entre mil y dos mil palabras) y la entrada puede tener numerosos formatos diferentes. Por ejemplo, usted puede entrar una instrucción directa (como Beber la cerveza), una instrucción múltiple (Coger el paquete y después guardárselo en el bolsillo), o una pregunta directa (¿Dónde estoy?).

El jugador se puede dirigir a los personajes simplemente diciendo sus nombres, como en Ford, ¿dónde estamos? o Marvin, vete. Incluso aunque el ordenador no comprenda exactamente lo que se espera de él, el programa por lo general aparecerá con una respuesta aceptable: una enorme mejora respecto a respuestas tradicionales como No puedes hacer eso o, simplemente, No comprendo.

Otras técnicas de programación incorporadas en este juego incluyen la provisión de «contenedores» (objetos que pueden retener otros objetos), una facilidad que con frecuencia falta en los juegos europeos. Otras provisiones son «objetos globales» (tales como «suelo», «pared», etc.), que pueden estar presentes en muchos escenarios diferentes, y «vehículos», contenedores que pueden retener y transportar al jugador de un escenario a otro. Es interesante observar, no obstante, que la mayoría de estas facilidades, junto con el enorme vocabulario y el eficaz analizador gramatical, son resultado del mayor almacenamiento de datos permitido por los discos y no de una programación más inteligente.

El desensamblado de un juego producido por Infocom revela un diseño muy complejo del argumento y el guión, pero poco en cuanto a lo que hemos dado en llamar inteligencia artificial. Entre algunos programadores de aventuras europeos existe la creencia generalizada de que la exuberancia de posibilidades que ofrece la programación basada en disco ha vuelto algo perezosos a sus colegas norteamericanos y que, cuando el mercado europeo abandone las cassettes, el hábito de aplicar las técnicas de compresión desarrolladas para los juegos basados en RAM traerá como lógica consecuencia la producción de programas de calidad superior.

Editorial Delta, S.A.



9

MI COMPUTER

Editorial Delta

10

MI COMPUTER

Editorial Delta

# MI COMPUTER SE PROLONGA

Con la aparición del fascículo 96 **"MI COMPUTER**, Curso Práctico del ordenador personal, el micro y el miniordenador" llega a su fin previsto, tras casi dos años de cita obligada y agradecida con nuestros lectores.

Pero el mundo de la informática está en contínua evolución y para responder al interés demostrado por nuestros lectores y a las numerosas cartas recibidas, hemos decidido enriquecer esta obra con **DOS NUEVOS VOLUMENES**, de 12 fascículos cada uno.

A lo largo de estos 24 nuevos fascículos, los lectores encontrarán la información más completa y actual sobre los últimos avances tecnológicos en el campo de la informática.

**MI COMPUTER** les ha introducido en el complejo mundo de la informática; la prolongación de la obra le convertirá en un experto.

